8th Workshop on Process Algebra and
Stochastically Timed Activities

PASTA 2009

PASTA: Wednesday 26th August 2009
Bio-PASTA: Thursday 27th August 2009

Informatics Forum
University of Edinburgh
UK

Preface

Welcome to the proceedings of PASTA and Bio-PASTA 2009. This is our favourite scientific meeting of the year bringing together established researchers, early career researchers and PhD students all intent on a free and frank exchange of important ideas and results. The nature of the group of attendees at the average PASTA workshop has always been conducive to just such a well-focused forum as each are able to understand and appreciate each others' work. In addition PASTA has a proven record of fostering successful collaboration efforts from across institutes.

This is the eighth annual meeting for collaborators interested in stochastic modelling. Continuing from last year, the workshop is now organised as two themed days; one for researchers interested in biological modelling and the specific challenges involved therein, and the other for researchers with a more general interest in modelling through stochastic process algebras.

As with every year our mission at the PASTA workshop is to have as much fun as we possibly can with a light sprinkling of scientific advancement. This year we are competing with the attractions of the Edinburgh Festival and we hope this will add to the enjoyment. Thanks for attending PASTA.

Special thanks to the Center for Systems Biology in Edinburgh (CSBE) and the Scottish Informatics and Computer Science Alliance (SICSA) for supporting the workshop.

Allan Clark and Maria Luisa Guerriero
26th August 2009

Contents

Part I

PASTA

The hybrid way to fluid-flow approximation — Invited Talk —

Luca Bortolussi
University of Trieste, Italy

Abstract

Discreteness and continuity are two opposite ways of describing the world. However, there are many situations, from engines to cells, in which an hybrid description, mixing both such ingredients, seems more adequate. In this presentation, we focus on stochastic hybrid systems. We present Piecewise Deterministic Markov Processes, and we provide a simpler description language, Transition-Driven Stochastic Hybrid Automata. Then, we will see how such formalism can be used to define in a simple way a stochastic hybrid semantics for HYPE (an hybrid process algebra focussing on flows) and for PEPA. Finally, we discuss convergence theorems for the PEPA case.

Differential Analysis of PEPA Models*

Mirco Tribastone†

1 Introduction

Continuous-time Markov chains (CTMCs) are an established tool for the quantitative evaluation of systems, however the well-known problem of state-space explosion makes the analysis intractable when the population sizes of the components under study are large. An alternative technique for performance evaluation may be offered by *deterministic models*, which use ordinary differential equations (ODEs) as the underlying mathematical structure, approximating the temporal evolution of the population of inherently discrete entities in a continuous fashion. Despite their apparently contrasting modelling approach, in many circumstances it is possible to establish a very useful relationship of convergence between the stochastic and deterministic models, where the ODE is interpreted as the fluid limiting behaviour of a *family* of CTMCs associated with the model under evaluation and parametrised by a system factor such as density or concentration [1].

The main contribution of this paper is to demonstrate that this result of deterministic convergence holds for population models described with the process algebra PEPA [2]. This objective is pursued by developing an operational semantics of the language which gives rise to a compact symbolic representation of the CTMC of the model, from which it is possible to infer the corresponding ODE representing its fluid limit. This semantics provides a formal account of earlier approaches to deterministic interpretations of PEPA [3], and substantially extends their scope of applicability by incorporating all the operators of the language and removing earlier assumptions on the syntactical structure of the models amenable to this analysis.

2 Population-based Semantics

The following example will be used throughout this section in order to illustrate the rationale behind the population-based semantics of PEPA.

Example 1 (PEPA model with cooperation)**.**

$$
\begin{array}{lrcl}
(\xi_{1,1}) & P & \stackrel{\text{def}}{=} & (\alpha, p).P' \\
(\xi_{1,2}) & P' & \stackrel{\text{def}}{=} & (\beta, p').P \\
(\xi_{2,1}) & Q & \stackrel{\text{def}}{=} & (\alpha, q).Q' \\
(\xi_{2,2}) & Q' & \stackrel{\text{def}}{=} & (\gamma, q').Q \\
 & \mathit{System}_1 & \stackrel{\text{def}}{=} & P[N_P] \underset{\{\alpha\}}{\bowtie} Q[N_Q]
\end{array}
$$

*Extended abstract

†School of Informatics, The University of Edinburgh, Scotland, UK. Email: mtribast@inf.ed.ac.uk

The *reduced context* of a PEPA model abstracts away from the actual population levels, considering one representative sequential component in place of a parallel composition of identical (i.e., isomorphic) components. In the running example, the reduced context is

$$\mathcal{M} \stackrel{def}{=} P \underset{\{\alpha\}}{\bowtie} Q.$$

This minimal form contains the necessary information to determine the state descriptor in the *numerical vector form* (NVF).

Definition 1 (Numerical Vector Form)**.** *Let N_C be the number of distinct sequential components in $\mathcal{M}$. Let $\mathcal{C}_i$ be the derivative set of the i-th component, $i = 1, 2, \ldots, N_C$ and let N_i be its size, i.e., $N_i = |\mathcal{C}_i|$. Let $C_{i,j}$ denote the j-th derivative of the i-th component, $j = 1, 2, \ldots, N_i$. The state descriptor in the NVF, denoted by $\xi \in \mathbb{Z}^d, d = \sum_{i=1}^{N_C} N_i$, assigns a coordinate, denoted by $\xi_{i,j}$, to each local derivative $C_{i,j}$ and indicates the number of copies in the system which exhibit that derivative.*

Definition 2 (Initial State of the CTMC)**.** *The initial state of the CTMC is denoted by $\delta \in \mathbb{Z}^d$ and gives an initial population level $\delta_{i,j} \geq 0$ to each local derivative $C_{i,j}$. Without loss of generality we exclude the case in which all the derivatives of a sequential component are set to 0, by subjecting δ to the condition $\sum_{k=1}^{N_i} \delta_{i,k} > 0$, for all i.*

The coordinates in the NVF of the sequential components in Example 1 are in parenthesis alongside the definitions.

As with the Markovian interpretation, at the core of this semantics is the notion of apparent rate. Here this concept is modified to take into account the interpretation of the reduced context described above.

Definition 3 (Parametric Apparent Rate)**.** *Consider a process P composed by sequential components $C_{i,j}$. The* parametric apparent rate *of action type α in component P, denoted by $r_\alpha^\star(P, \xi)$, defines the overall rate at which the action type α can be performed by component P as a function of the population sizes ξ of the sequential components of the system:*

$$r_\alpha^\star\left(P \underset{L}{\bowtie} Q, \xi\right) = \begin{cases} \min\left(r_\alpha^\star(P,\xi), r_\alpha^\star(Q,\xi)\right) & \text{if } \alpha \in L \\ r_\alpha^\star(P,\xi) + r_\alpha^\star(Q,\xi) & \text{if } \alpha \notin L \end{cases}$$

$$r_\alpha^\star(P/L, \xi) = \begin{cases} r_\alpha^\star(P,\xi) & \text{if } \alpha \notin L \\ 0 & \text{if } \alpha \in L \end{cases}$$

$$r_\alpha^\star(C_{i,j}, \xi) = \sum_{k=1}^{N_i} r_\alpha(C_{i,k})\xi_{i,k}$$

In the last definition, the behaviour of the other derivatives in the same derivative set of $C_{i,j}$ is taken into account because each constituting sequential component in the reduced context in fact represents an array of identical components, evolving through the local derivatives $C_{i,k}$, $1 \leq k \leq d$. In any state of the CTMC there may be at least one component exhibiting each such derivative. These components will *compete* for the execution of some shared action α, and the propensity of each derivative will be proportional to the population level of the derivative and the individual rate of execution.

The population-based structured operational semantics for PEPA is shown in Table 1. The rule for sequential components $\mathrm{S}_0^\star$ constructs the relationship between the two semantics. The premise is a transition of the Markovian semantics for a single sequential component. By construction, the right hand side of the transition is in the same derivative set, i.e., $C_{i,j} \xrightarrow{(\alpha,r)} C_{i',j'} \Rightarrow i = i'$. Such a transition is said to be *promoted* to an inference

Table 1: Population-based parametric structured operational semantics of PEPA. Transitions are denoted by the symbol $\longrightarrow_\star$ to distinguish them from the Markovian transitions in PEPA which carry reals instead of functions.

Sequential Component (Promotion Rule)

$$\mathrm{S}_0^\star : \frac{C_{i,j} \xrightarrow{(\alpha,r)} C_{i,j'} \quad C_{i,j} \in \mathcal{C}_i}{C_{i,j} \xrightarrow{(\alpha, r\xi_{i,j})}_\star C_{i,j'}}$$

Constant

$$\mathrm{A}_0^\star : \frac{P \xrightarrow{(\alpha,r(\xi))}_\star P'}{A \xrightarrow{(\alpha,r(\xi))}_\star P'}, \; A \stackrel{def}{=} P$$

Cooperation

$$\mathrm{C}_0^\star : \frac{P \xrightarrow{(\alpha,r(\xi))}_\star P'}{P \underset{L}{\bowtie} Q \xrightarrow{(\alpha,r(\xi))}_\star P' \underset{L}{\bowtie} Q}, \; \alpha \notin L \qquad \mathrm{C}_1^\star : \frac{Q \xrightarrow{(\alpha,r(\xi))}_\star Q'}{P \underset{L}{\bowtie} Q \xrightarrow{(\alpha,r(\xi))}_\star P \underset{L}{\bowtie} Q'}, \; \alpha \notin L$$

$$\mathrm{C}_2^\star : \frac{P \xrightarrow{(\alpha,r_1(\xi))}_\star P' \;\; Q \xrightarrow{(\alpha,r_2(\xi))}_\star Q'}{P \underset{L}{\bowtie} Q \xrightarrow{(\alpha,r(\xi))}_\star P' \underset{L}{\bowtie} Q'}, \; \alpha \in L,$$

$$r(\xi) = \frac{r_1(\xi)}{r_\alpha^\star(P,\xi)} \frac{r_2(\xi)}{r_\alpha^\star(Q,\xi)} \min\left(r_\alpha^\star(P,\xi), r_\alpha^\star(Q,\xi)\right)$$

Hiding

$$\mathrm{H}_0^\star : \frac{P \xrightarrow{(\alpha,r(\xi))}_\star P'}{P/L \xrightarrow{(\alpha,r(\xi))}_\star P'/L}, \; \alpha \notin L \qquad \mathrm{H}_1^\star : \frac{P \xrightarrow{(\alpha,r(\xi))}_\star P'}{P/L \xrightarrow{(\tau,r(\xi))}_\star P'/L}, \; \alpha \in L$$

for the population-based semantics — the premise describes the behaviour of a single sequential component, whereas the conclusion gives the collective dynamics of the population of components $C_{i,j}$. This population evolves at an overall rate which is the product of the individual rate and the number of components exhibiting this local derivative. The other rules are syntactically similar to their counterparts in the Markovian semantics, however in all cases the derivations carry as rates functions of the population vector. The following derivation tree gives a transition for the shared activity with regard to the reduced context of Example 1.

$$\frac{\dfrac{P \xrightarrow{(\alpha,p)} P'}{P \xrightarrow{(\alpha,p\xi_{1,1})}_\star P'}\,\mathrm{S}_0^\star \qquad \dfrac{Q \xrightarrow{(\alpha,q)} Q'}{Q \xrightarrow{(\alpha,q\xi_{2,1})}_\star Q'}\,\mathrm{S}_0^\star}{P \underset{\{\alpha\}}{\bowtie} Q \xrightarrow{(\alpha,\min(p\xi_{1,1},q\xi_{2,1}))}_\star P' \underset{\{\alpha\}}{\bowtie} Q'}\,\mathrm{C}_2^\star$$

However, this information is not sufficient to obtain the behaviour of the entire system under consideration, because the derivative gives only the first-step behaviour of the process. The collective dynamics of the system is represented by the notions of derivative set and derivation graph of $\mathcal{M}$ in the population-based semantics, which are defined in a similar way to their counterparts in the Markovian semantics.

Definition 4 (Parametric Derivative Set)**.** *The* parametric derivative set *of $\mathcal{M}$, denoted by $ds^\star(\mathcal{M})$, is the smallest set of PEPA components which satisfies the following conditions:*

- $\mathcal{M} \in ds^\star(\mathcal{M})$
- *If* $P \in ds^\star(\mathcal{M})$ *and there exists* $P \xrightarrow{(\alpha, r(\xi))}_\star P'$ *then* $P' \in ds^\star(\mathcal{M})$

Definition 5 (Parametric Derivation Graph)**.** *Given a parametric derivative set* $ds^\star(\mathcal{M})$*, the* parametric derivation graph *of* $\mathcal{M}$*, denoted by* $\mathcal{D}^\star(\mathcal{M})$ *is a labelled directed multi-graph* (V, A) *with vertices* $V \in ds^\star(\mathcal{M})$ *and arcs* $A \in ds^\star(\mathcal{M}) \times \mathcal{L} \times ds^\star(\mathcal{M})$ *where the number of occurrences of an arc is equal to the number of distinct inference trees for a transition.*

The transitions of $\mathcal{D}^\star(\mathcal{M})$ in Example 1 are:

$$P \underset{\{\alpha\}}{\bowtie} Q \xrightarrow{(\alpha, \min(p\xi_{1,1}, q\xi_{2,1}))}_\star P' \underset{\{\alpha\}}{\bowtie} Q'$$

$$P' \underset{\{\alpha\}}{\bowtie} Q' \xrightarrow{(\beta, p'\xi_{1,2})}_\star P \underset{\{\alpha\}}{\bowtie} Q'$$

$$P' \underset{\{\alpha\}}{\bowtie} Q' \xrightarrow{(\gamma, q'\xi_{2,2})}_\star P' \underset{\{\alpha\}}{\bowtie} Q$$

$$P \underset{\{\alpha\}}{\bowtie} Q' \xrightarrow{(\gamma, q'\xi_{2,2})}_\star P \underset{\{\alpha\}}{\bowtie} Q$$

The parametric derivation graph leads to a set of *generating functions* of the CTMC, i.e., functions of the state descriptor which give the transition rates to all the reachable states of the system. These functions are parametrised by action types to keep track of the additional information about which action type is associated with a transition. Let $l \in \mathbb{Z}^d$ denote a *transition jump*, recording the changes in the population levels of ξ due to a transition of the chain. The generating functions are denoted by $\varphi_\alpha(\xi, l) : \mathbb{R}^d \to \mathbb{R}$ and give the transition rate for a jump l and an activity of type α. Thus, the entry in the generator matrix corresponding to the transition from ξ to $\xi + l$, denoted by $q_{\xi,\xi+l}$, can be written as

$$q_{\xi,\xi+l} = \sum_{\alpha \in \mathcal{A}} \varphi_\alpha(\xi, l).$$

The summation across $\mathcal{A}$ captures the fact that distinct action types may contribute to a transition to the same target state, e.g., $(\alpha, p).P + (\beta, s).P$.

A transition jump is calculated using the notion of *indicator function*, which gives the coordinate in the NVF of the components in the parametric derivation graph of a model.

Definition 6 (Indicator Function)**.** *Let* $1_{i,j} \in \mathbb{Z}^d$ *denote a vector whose elements are all zero except for the coordinate corresponding to the derivative* $C_{i,j}$*, which is set to one. Let* $P \in ds^\star(\mathcal{M})$*. The* indicator *of* P*, denoted by* $ind(P)$*, returns a vector whose non-zero elements correspond to the indices in the population vector of the sequential components in* P*. It is defined as follows:*

$$\begin{aligned} ind(C_{i,j}) &= 1_{i,j} \\ ind(A \stackrel{\text{def}}{=} P) &= ind(P) \\ ind(P \underset{L}{\bowtie} Q) &= ind(P) + ind(Q) \\ ind(P/L) &= ind(P)/L \end{aligned}$$

For instance, the indicator of $P \underset{\{\alpha\}}{\bowtie} Q$ is $l_{1,1} = 1$, $l_{1,2} = 0$, $l_{2,1} = 1$, $l_{2,2} = 0$. For a transition in the parametric derivation graph, the jump is calculated as the difference between the indicator of the lhs and that of the rhs of the transition. For instance, the first transition has a jump with coordinates $l_{1,1} = -1$, $l_{1,2} = 1$, $l_{2,1} = -1$, $l_{2,2} = 1$, which correctly gives the changes in the population levels due to the execution of the shared transition α. The complete set of generating functions is computed according to the following definition.

Definition 7 (Extraction of the Generating Functions)**.** *Let $\mathcal{M}$ be a PEPA model with parametric derivative graph $\mathcal{D}^\star(\mathcal{M})$. The generating functions of the underlying population-based CTMC are as follows:*

$$\varphi_\alpha(\xi, l) = \begin{cases} m \cdot r(\xi) & \text{if } \exists\ P \xrightarrow{(\alpha, r(\xi))}_\star P' \in A \text{ and } l = 0^d - ind(P) + ind(P') \\ 0 & \text{otherwise} \end{cases}$$

where m is the arc multiplicity and 0^d is the zero-vector in $\mathbb{Z}^d$.

For instance, the generating function implied by $P \underset{\{\alpha\}}{\bowtie} Q \xrightarrow{(\alpha, \min(p\xi_{1,1}, q\xi_{2,1}))}_\star P' \underset{\{\alpha\}}{\bowtie} Q'$ is $\varphi_\alpha(\xi, l) = \min(p\xi_{1,1}, q\xi_{2,1})$, where l is the jump vector discussed above.

Finally, a vector field $F(x)$ can be inferred from the generating functions of a PEPA model as

$$F(x) = \sum_{l \in \mathbb{Z}^d} l \sum_\alpha \varphi_\alpha(x, l) \tag{1}$$

and the underlying ODE is defined as

$$\frac{d}{dt} x(t) = F(x(t)). \tag{2}$$

In components, the ODE underlying Example 1 is:

$$\begin{aligned} \dot{x}_{1,1} &= -\min(p\,x_{1,1}, q\,x_{2,1}) + p'\,x_{1,2} \\ \dot{x}_{1,2} &= \min(p\,x_{1,1}, q\,x_{2,1}) - p'\,x_{1,2} \\ \dot{x}_{2,1} &= -\min(p\,x_{1,1}, q\,x_{2,1}) + q'\,x_{2,2} \\ \dot{x}_{2,2} &= \min(p\,x_{1,1}, q\,x_{2,1}) - q'\,x_{2,2} \end{aligned}$$

This formulation makes it possible to establish a property of convergence for PEPA models according to the interpretation by Kurtz [1], [4]. The result used here states that the solution to a properly defined initial value problem with (2) is the fluid limiting behaviour of a family of CTMCs in the sense of the following theorem.

Theorem 1 (cfr. [1], Theorem 3.1)**.** *Let $\{X_n(t)\}$ be a family of* density dependent *CTMCs, i.e., a sequence of chains with parameter $n \in \mathbb{N}$ taking values in $\mathbb{Z}^d$ such that the infinitesimal generator entries for $X_n(t)$, denoted by $q_{\xi,\xi+l}$, can be described as*

$$q_{\xi,\xi+l} = n \cdot \varphi(\xi/n, l). \tag{3}$$

Suppose that:

1. *The functions $\varphi(x, l)$ are continuous.*
2. *There exists an open set $E \subset \mathbb{R}^d$ and a constant $L_E \in \mathbb{R}$ such that:*
   - *(a)* $\|F(x) - F(y)\| < L_E \|x - y\|, \quad x, y \in E$
   - *(b)* $\sup_{x \in E} \sum_{l \in \mathbb{Z}^d} \|l\| \varphi(x, l) < \infty$
   - *(c)* $\lim_{k \to \infty} \sup_{x \in E} \sum_{\|l\| > k} \|l\| \varphi(x, l) = 0$

Then, for every solution to the initial value problem of (2) subject to

$$x(0) = x_0 \quad \text{and} \quad x(t) \in E, \quad 0 \le t \le T$$

the family $\{X_n(t)\}$ converges to $x(t)$ in the sense that

$$\lim_{n \to \infty} X_n(0)/n = \delta \Longrightarrow \forall \varepsilon > 0 \lim_{n \to \infty} \mathbb{P}\left(\sup_{t \le T} \|X_n(t)/n - x(t)\| > \varepsilon \right) = 0.$$

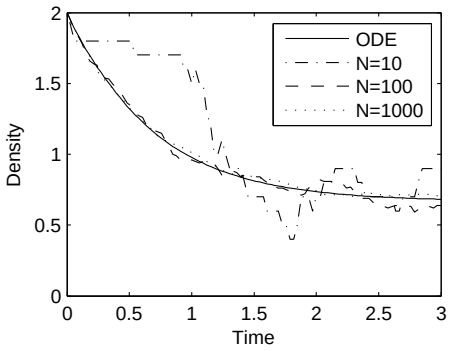


Figure 1: Density of component P in Example 1. One realisation of the scaled Markov chain $X_n(t)/n$ over the first three time units becomes closer to the solution of the ODE as n increases. Parameter set: $p = 1.0, p' = 0.5, q = 2.0, q' = 4.0, \delta = (2, 0, 1, 0)$.

3 Practical Implications

A family of CTMCs which satisfy $\lim_{n\to\infty} X_n(0)/n = \delta$ can be taken by letting the initial population levels be multiples of δ, i.e. $X_n(0) = n\delta$ for all n. This corresponds to increasingly large state spaces as a function of n. For instance $X_1(t)$ is the CTMC with the system equation in Example 1, $X_2(t)$ corresponds to the initial state $P[2N_P] \underset{\{\alpha\}}{\bowtie} Q[2N_Q]$, and so on. The result of convergence intuitively states that, asymptotically, a sample path of the CTMC $X_n(t)$ may be well approximated by $n \cdot x(t)$, over any finite time interval, where $x(t)$ is the solution to the initial value problem of the ODE with $x(0) = \delta$. A pictorial representation of this result is given in Fig. 1, which shows that the ODE is a closer approximation to sample paths of $X_n(t)/n$ for increasingly large n, with excellent accuracy at $n = 1000$.

In conclusion, the results presented in this paper give confidence on the soundness of the differential analysis of a large class of PEPA models (the result of convergence does not hold for models with passive actions because of the discontinuity of the generating functions for expressing unbounded capacity). This approach is particularly suitable for large-scale population models, easily susceptible to combinatorial growth of the state space when expressed with discrete-state techniques, providing accurate solutions with little computational cost.

Acknowledgements The author is supported by the EU-funded project SENSORIA, IST-2005-016004. Numerous discussions with Jane Hillston and Stephen Gilmore have helped shape this work.

References

[1] T. G. Kurtz. Solutions of ordinary differential equations as limits of pure Markov processes. *J. Appl. Prob.*, 7(1):49–58, April 1970.

[2] J. Hillston. *A Compositional Approach to Performance Modelling*. Cambridge University Press, 1996.

[3] J. Hillston. Fluid flow approximation of PEPA models. In *Proceedings of the Second International Conference on the Quantitative Evaluation of Systems*, pages 33–43, Torino, Italy, September 2005. IEEE Computer Society Press.

[4] T. G. Kurtz. Limit theorems for sequences of jump Markov processes approximating ordinary differential processes. *J. Appl. Prob.*, 8(2):344–356, 1971.

A functional central limit theorem for PEPA

Richard A. Hayden Jeremy T. Bradley

Dept. of Computing, Imperial College London
180 Queen's Gate, London SW7 2BZ, UK

{rh,jb}@doc.ic.ac.uk

August 19, 2009

Abstract

We present a functional central limit theorem which quantifies, as a stochastic process, the difference between a PEPA model's underlying CTMC and its fluid approximation. We generalise existing theory to handle the case of non-smooth rate functions, which is an issue particular to modelling problems in computer science. We verify the weak convergence empirically and suggest future avenues for deducing more analytic approximations from it.

1 Introduction

Fluid-analysis of performance models offers the exciting potential of analysing massive state spaces at small computational cost. In the case of stochastic process algebra models, fluid-analysis involves approximating the underlying discrete state space with continuous real-valued variables and describing the time-evolution of those variables with ordinary differential equations (ODEs). This approach was first applied to a subset of the stochastic process algebra PEPA [1] by Hillston [2], have since been extended and developed in a number of different directions in the literature [3; 4; 5]. Furthermore, similar ideas have been applied in other stochastic process algebra [6; 7] and stochastic Petri net [8] formalisms.

Despite the successful and widespread application of these techniques, see e.g. [9; 3; 10; 11], many questions still exist regarding the relationship of the approximation to the original stochastic model — its underlying continuous time Markov chain (CTMC). In this paper, we explore one avenue for better understanding the relationship, a functional central limit theorem, which quantifies second-order deviations from the first-order fluid approximation. The result is a continuous state space stochastic process, which lies between the fluid approximation and the underlying CTMC in terms of accuracy and tractability.

In the following section, Section 1.1, we introduce the stochastic process algebra PEPA and in Section 1.2, we introduce the fluid semantics by means of a simple client/server model for the sake of brevity. In Section 2, we present the functional central limit theorem and some examples, again in terms of the client/server model. Then, Section 2.2 discusses the representation of the limit process as the solution to a stochastic differential equation and the subsequent derivation of its Fokker-Planck partial differential equation. Finally, we conclude in Section 3.

1.1 PEPA

PEPA [1; 12] as a performance modelling formalism has been used to study a wide variety of systems, including multimedia applications [13], mobile phone usage [14], GRID scheduling [15], production cell efficiency [16] and web-server clusters [17] amongst others. It is also adept at capturing large parallel

software systems, such as peer-to-peer networks [9], to which the style of analysis considered here is particularly suited.

As in all process algebras, systems are represented in PEPA as the composition of *components* which undertake *actions*. In PEPA the actions are assumed to have a duration, or delay. Thus the expression $(\alpha, r).P$ denotes a component which can undertake an α action at rate r to evolve into a component P. Here $\alpha \in \mathcal{A}$ where $\mathcal{A}$ is the set of action types. The rate r is interpreted as a random delay which samples from an exponential random variable with parameter, r.

PEPA has a small set of combinators, allowing system descriptions to be built up as the concurrent execution and interaction of simple sequential components. The syntax of the type of PEPA model considered in this paper may be formally specified using the following grammar:

$$\begin{aligned} S &::= (\alpha, r).S \mid S + S \mid C_S \\ P &::= P \underset{L}{\bowtie} P \mid P/L \mid C \end{aligned}$$

where S denotes a *sequential component* and P denotes a *model component* which executes in parallel. C stands for a constant which denotes either a sequential component or a model component as introduced by a definition. C_S stands for constants which denote sequential components. The effect of the syntactic separation between these types of constants is to constrain legal PEPA components to be cooperations of sequential processes.

More information and structured operational semantics on PEPA can be found in [1]. A brief discussion of the basic PEPA operators is given below:

Prefix The basic mechanism for describing the behaviour of a system with a PEPA model is to give a component a designated first action using the prefix combinator, denoted by a full stop, which was introduced above. As explained, $(\alpha, r).P$ carries out an α action with rate r, and it subsequently behaves as P.

Choice The component $P + Q$ represents a system which may behave either as P or as Q. The activities of both P and Q are enabled. The first activity to complete distinguishes one of them: the other is discarded. The system will behave as the derivative resulting from the evolution of the chosen component.

Constant It is convenient to be able to assign names to patterns of behaviour associated with components. Constants are components whose meaning is given by a defining equation. The notation for this is $X \stackrel{def}{=} E$. The name X is in scope in the expression on the right hand side meaning that, for example, $X \stackrel{def}{=} (\alpha, r).X$ performs α at rate r forever.

Hiding The possibility to abstract away some aspects of a component's behaviour is provided by the hiding operator, denoted P/L. Here, the set L identifies those activities which are to be considered internal or private to the component and which will appear as the unknown type τ.

Cooperation We write $P \underset{L}{\bowtie} Q$ to denote cooperation between P and Q over L. The set which is used as the subscript to the cooperation symbol, the *cooperation set* L, determines those activities on which the components are forced to synchronise. For action types not in L, the components proceed independently and concurrently with their enabled activities. We write $P \parallel Q$ as an abbreviation for $P \underset{L}{\bowtie} Q$ when L is empty. Furthermore, $P[n]$ is shorthand for the parallel cooperation of n P-components, $\underbrace{P \mid\mid \cdots \mid\mid P}_{n}$.

In process cooperation, if a component enables an activity whose action type is in the cooperation set it will not be able to proceed with that activity until the other component also enables an activity of that type. The two components then proceed together to complete the *shared activity*. Once enabled, the rate of a shared activity has to be altered to reflect the slower component in a cooperation.

In some cases, when a shared activity is known to be completely dependent only on one component in the cooperation, then the other component will be made *passive* with respect to that activity. This means that the rate of the activity is left unspecified (denoted $\top$) and is determined upon cooperation, by the rate of the activity in the other component. All passive actions must be synchronised in the final model.

Within the cooperation framework, PEPA respects the definition of *bounded capacity*: that is, a component cannot be made to perform an activity faster by cooperation, so the rate of a shared activity is the minimum of the apparent rates of the activity in the cooperating components.

1.2 First-order fluid analysis

For the sake of brevity, we will not formally present here the fluid semantics for PEPA. It can be found in different degrees of generality in the literature (e.g. [2; 18; 5]). Instead, we will introduce the techniques by considering a simple case study.

In the PEPA model $System$ below, we have a population of N_C Clients and a population of N_S Servers. The system uses a 2-stage fetch mechanism: a client requests data from the pool of servers; one of the servers receives the request, another server may then fetch the data for the client. At any stage, a server in the pool may fail.

$$\begin{aligned}
Client &\stackrel{def}{=} (request, r_{req}).Client_waiting \\
Client_waiting &\stackrel{def}{=} (data, r_{data}).Client_think \\
Client_think &\stackrel{def}{=} (think, r_{think}).Client \\
\\
Server &\stackrel{def}{=} (request, r_{req}).Server_get + (break, r_{break}).Server_broken \\
Server_get &\stackrel{def}{=} (data, r_{data}).Server + (break, r_{break}).Server_broken \\
Server_broken &\stackrel{def}{=} (reset, r_{reset}).Server \\
\\
System &\stackrel{def}{=} Client[N_C] \underset{L}{\bowtie} Server[N_S]
\end{aligned}$$

where $L = \{request, data\}$.

Since each client and server can be in one of three derivative states, it is clear that this model has $3^{N_C+N_S}$ states in its underlying CTMC, and thus it is quickly intractable to traditional analysis methods. Consider the three integer-valued stochastic processes which count the number of the N_C clients in each of the three possible derivative states of $Client$. Let these be $N_C(t)$, $N_{C_w}(t)$ and $N_{C_t}(t)$ respectively. Similarly, define for the servers, $N_S(t)$, $N_{S_g}(t)$ and $N_{S_b}(t)$. Using *strong equivalence* it is straightforward to show that the partition of the state space into mutually exclusive subsets, such that all of these stochastic processes take on the same value in each subset, is a *lumpable* partition, see [1, Chapter 8]. This allows these states to be combined and the rates aggregated, resulting in a smaller CTMC, for which each state is specified uniquely by the values of the six stochastic processes defined above. We call this the *underlying aggregated CTMC*. Unfortunately, this simplification does not, in general, solve the state space explosion problem.[1] However, it is a necessary first step for deriving differential equations to perform the fluid analysis.

The idea of the fluid-analysis is to define deterministic, real-valued fluid approximations $v.(t)$ (defined by ODEs) to the integer stochastic processes $N.(t)$. In order to construct the ordinary differential equation which governs the evolution of $v_C(t)$, for example, we consider the aggregate CTMC rate at which $Client$ components are lost in the model and the rate at which they are gained, balancing the two quantities in terms of the fluid approximations $v.(t)$:

$$\dot{v}_C(t) = -\min(v_C(t),\, v_S(t))r_{req} + v_{C_t}(t)r_{think} \tag{1.1}$$

[1] Indeed, the aggregated state space of this model consists of potentially $\frac{1}{4}(2 + N_c)(1 + N_c)(2 + N_s)(1 + N_s)$ states. For $N_c = 100$ and $N_s = 50$, this is $6,830,226$ states. In general, the size of the aggregated space grows quickly as the number of possible derivative states increases.

That is, $Client$ components are lost only through evolving into $Client\_waiting$ components. This happens by virtue of completing a $request$ shared action with a $Server$ component, at the aggregate CTMC rate $\min(N_C(t),\, N_S(t))r_{req}$. $Client$ components are gained only through $Client\_think$ components completing their $think$ action at aggregate CTMC rate $N_{C_t}(t)r_{think}$. Similar considerations for the other client and server components lead to a complete set of six ODEs. These can then be inexpensively integrated to obtain the $v_\cdot(t)$ as deterministic, real-valued functions.

In the case of a general PEPA model, assume that we have a vector-valued stochastic process $\mathbf{N}(t)$, defined on $\mathbb{R}_+$ taking values in $\mathbb{Z}_+^N \subset \mathbb{R}_+^N$, for some $N \in \mathbb{Z}_+$. Each component of this process counts the number of a particular derivative state currently active in a parallel group of the model, of which there are N derivative states in total, across all parallel groups.

Analogously, we define the vector-valued deterministic function $\mathbf{v}(t)$, also defined on $\mathbb{R}_+$ and taking values in $\mathbb{R}_+^n$ to be the first-order fluid approximation of this model. We assume that it is defined uniquely by the following system of differential equations:

$$\dot{\mathbf{v}}(t) = \mathbf{f}(\mathbf{v}(t))$$

and the initial condition $\mathbf{v}(0) = \mathbf{N}(0)$. That is, the deterministic function $\mathbf{f} : \mathbb{R}_+^N \to \mathbb{R}^N$ corresponds component-wise to the rate at which each derivative state is incremented, minus that at which it is decremented, in a given state of the model. In the case of the above example, we have:

$$\begin{aligned} n &= 6 \\ \mathbf{N}(t) &\equiv (N_C(t),\, N_{C_w}(t),\, N_{C_t}(t),\, N_S(t),\, N_{S_g}(t),\, N_{S_b}(t))^T \\ \mathbf{v}(t) &\equiv (v_C(t),\, v_{C_w}(t),\, v_{C_t}(t),\, v_S(t),\, v_{S_g}(t),\, v_{S_b}(t))^T \end{aligned}$$

and:

$$\mathbf{f}(\mathbf{v}(t)) \equiv \begin{pmatrix} -\min(v_S(t),\, v_C(t))r_{req} + v_{C_t}(t)r_{think} \\ -\min(v_{C_w}(t),\, v_{S_g}(t))r_{data} + \min(v_S(t),\, v_C(t))r_{req} \\ -v_{C_t}(t)r_{think} + \min(v_{C_w}(t),\, v_{S_g}(t))r_{data} \\ -\min(v_S(t),\, v_C(t))r_{req} - v_S(t)r_{break} + \min(v_{C_w}(t),\, v_{S_g}(t))r_{data} + v_{S_b}(t)r_{reset} \\ -\min(v_{C_w}(t),\, v_{S_g}(t))r_{data} - v_{S_g}(t)r_{break} + \min(v_S(t),\, v_C(t))r_{req} \\ -v_{S_b}(t)r_{reset} + v_{S_g}(t)r_{break} + v_S(t)r_{break} \end{pmatrix}$$

To see in more detail how an arbitrary PEPA model can be represented in such a manner, the reader is directed to [18]. Furthermore, it is clear that similar models in other related formalisms might also be cast into such a representation, thus extending the scope of this paper beyond just PEPA.

The following theorem, which can be proved using a result of Kurtz [19], is now relatively well known in the community. It gives a limiting convergence in probability relationship of the first-order fluid analysis to the underlying CTMC, over bounded intervals of time. This convergence occurs when the component counts are scaled by the population size and is thus asserting that the relative error of the first-order fluid analysis decays in the limit of large populations.

In order to state the theorem, consider a sequence of PEPA models which have the same structure of parallel component groups and differ only in terms of the size of the component populations within these groups. Furthermore, we require that they all have the same initial proportion of each component type in each case. Let $\{\mathbf{N}^i(t)\}_{i=1}^\infty$ be the associated stochastic counting processes in the notation above, and for each i, write $S_i := N_1^i(t) + \cdots + N_n^i(t)$ for the total component population of model i.[2] So our requirement of constant initial component type proportions is stated formally as:

$$\frac{N_k^i(0)}{S_i} = \frac{N_k^j(0)}{S_j} \text{ for all } i,\, j > 0 \text{ and } k \in \{1,\, \ldots,\, n\}$$

In the case of PEPA, it is relatively straightforward to see that for any $\mathbf{x} \in \mathbb{R}_+^N$, $\mathbf{f}(k\mathbf{x}) = k\mathbf{f}(\mathbf{x})$ for all $k \in \mathbb{R}_+$. Furthermore, since our sequence of PEPA models differ only in terms of the initial component

[2] This does not depend on t because the PEPA operational semantics preserve component populations.

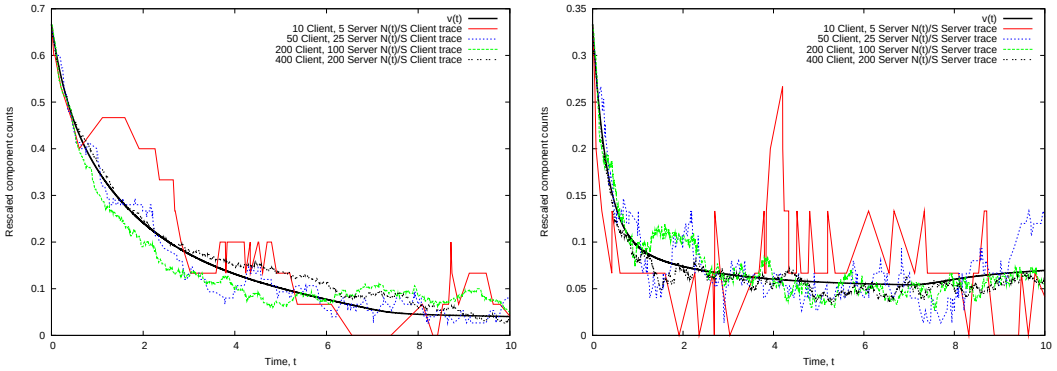


Fig. 1. Comparison of ODE approximation with scaled traces of the $Client$ and $Server$ counting processes for the client/server model. Rates used are $r_{req} = 3.0$, $r_{think} = 0.3$, $r_{break} = 0.3$, $r_{data} = 1.0$ and $r_{reset} = 0.2$.

counts, it is easy to see that the function $\mathbf{f}(\cdot)$ is the same for any i. These two facts together mean that we need only define the fluid approximation to $\mathbf{N}^i(t)$, say, $\mathbf{v}^i(t)$ for a particular value of i, and the fluid approximation for any other i can be defined in terms of it. Indeed, for any i, $j > 0$, if $\dot{\mathbf{v}}^i(t) = \mathbf{f}(\mathbf{v}^i(t))$ and $\dot{\mathbf{v}}^j(t) = \mathbf{f}(\mathbf{v}^j(t))$ with initial conditions, $\mathbf{v}^i(0) = \mathbf{N}^i(0)$ and $\mathbf{v}^j(0) = \mathbf{N}^j(0)$, respectively, we have:

$$\mathbf{v}^i(0) = \mathbf{v}^j(0) \times \frac{S_i}{S_j} \text{ and } \mathbf{v}^i(t) = \mathbf{v}^j(t) \times \frac{S_i}{S_j} \text{ for all } t > 0$$

Thus we choose only to consider the quantity $\mathbf{v}(t) := \mathbf{v}^i(t)/S_i$, for all $t > 0$, which we have just seen is independent of i. The theorem can then be stated in terms of these quantities as follows.

Theorem 1.1 *If $S_i \to \infty$ as $i \to \infty$, then, for all $\delta > 0$ and $T > 0$:*

$$\mathbb{P}\left\{ \sup_{t \in [0,T]} \left\| \mathbf{N}^i(t)/S_i - \mathbf{v}(t) \right\| > \delta \right\} \longrightarrow 0$$

as $i \to \infty$.

Proof. See Kurtz [19]. □

Figure 1 shows the effect of this theorem for the client/server example.

2 Second-order approximation (FCLT)

The purpose of this section is to improve upon the fluid approximation of a PEPA model's underlying CTMC by constructing a second-order stochastic approximation to the deviation of the CTMC from the fluid limit. An approach is given in another paper by Kurtz [20], but cannot be applied directly since we are working with non-smooth rate functions.

However, we will show in this section that we can generalise the theory so that a second-order stochastic limit does hold, at least as long as the first-order fluid approximation is sufficiently well-behaved. The proof will draw heavily on results found in [21, Chapters 6, 7 & 11].

It is worth mentioning here that the presence of non-smoothness in the rate functions is not unique to PEPA. Indeed, non-smooth rate functions are also found in continuous stochastic Petri nets [8] and it would appear that the minimum function in particular is the natural method of modelling synchronisation in the field of

computation. It is therefore important that we are able to deal with non-smooth functions when constructing stochastic approximations.

In the next section, we illustrate an alternative representation of a PEPA model, which will lead more naturally to the second-order stochastic approximation.

2.1 Random time change representation of PEPA models

The operational semantics of PEPA [1] specify the underlying CTMC (both aggregated and unaggregated) for a given model. In particular, this is normally achieved through the specification of the instantanenous transition rates between states. In this section, we illustrate an alternative representation of the aggregated CTMC in terms of simple stochastic primitives, which leads more readily to the stochastic limit of interest.

We will work in the same framework as before with our sequence of structurally identical PEPA models, with underlying aggregated CTMCs, $\{\mathbf{N}^i(t)\}_{i=1}^{\infty}$ and fluid approximation, $\mathbf{v}(t)$. However, we must now consider the transitions in the aggregated state space individually. For example, in the case of the client/server model, the transitions in the aggregated state space can be enumerated as:

1. $request$-transitions of one $Client$ to one $Client\_waiting$ and one $Server$ to one $Server\_get$ at rate $\min(N_C(t),\, N_S(t))r_{req}$,
2. $data$-transitions of one $Client\_waiting$ to one $Client\_think$ and one $Server\_get$ to one $Server$ at rate $\min(N_{C_w}(t),\, N_{S_g}(t))r_{data}$,
3. $think$-transitions of one $Client\_think$ to one $Client$ at rate $N_{C_t}(t)r_{think}$,
4. $break$-transitions of one $Server$ to one $Server\_broken$ at rate $N_S(t)r_{break}$,
5. $break$-transitions of one $Server\_get$ to one $Server\_broken$ at rate $N_{S_g}(t)r_{break}$,
6. $reset$-transitions of one $Server\_broken$ to one $Server$ at rate $N_{S_b}(t)r_{reset}$

In the more general case, we might represent the K transitions in the aggregated state space by a sequence of jump vectors, $\{\mathbf{l}^k \in \mathbb{Z}^N\}_{k=1}^K$ specifying that if the kth such transition occurs at time t, $\mathbf{N}(t) = \mathbf{N}(t-) + \mathbf{l}^k$, and a sequence of rate functions, $\{f^k : \mathbb{R}_+^N \to \mathbb{R}_+\}_{k=1}^K$, specifying the aggregate rate of each transition. For the example above ($K = 6$) and in that order of enumeration, we would have:

$$\begin{aligned}
\mathbf{l}^1 &= (-1,\, 1,\, 0,\, -1,\, 1,\, 0) & f^1(\mathbf{x}) &= \min(x_1,\, x_4)r_{req} \\
\mathbf{l}^2 &= (0,\, -1,\, 1,\, 1,\, -1,\, 0) & f^2(\mathbf{x}) &= \min(x_2,\, x_5)r_{data} \\
\mathbf{l}^3 &= (1,\, 0,\, -1,\, 0,\, 0,\, 0) & f^3(\mathbf{x}) &= x_3 r_{think} \\
\mathbf{l}^4 &= (0,\, 0,\, 0,\, -1,\, 0,\, 1) & f^4(\mathbf{x}) &= x_4 r_{break} \\
\mathbf{l}^5 &= (0,\, 0,\, 0,\, 0,\, -1,\, 1) & f^5(\mathbf{x}) &= x_5 r_{break} \\
\mathbf{l}^6 &= (0,\, 0,\, 0,\, 1,\, 0,\, -1) & f^6(\mathbf{x}) &= x_6 r_{reset}
\end{aligned}$$

We now consider a probability space equipped with K mutually independent standard (rate 1) Poisson processes, say $\{P^k(t)\}_{k=1}^K$, with the intention that $P^k(t)$ corresponds to transition k. It can then be shown that we may represent $\mathbf{N}^i(t)$ on the same probability space as the unique (in terms of sample-paths) solution to the following equation:

$$\mathbf{N}^i(t) = \mathbf{N}^i(0) + \sum_{k \in K} P^k\left(\int_0^t f^k(\mathbf{N}^i(s))\, ds\right)\mathbf{l}^k$$

It is also true that defining the CTMC, $\mathbf{N}^i(t)$ in this manner is equivalent in distribution to the usual instantaneous transition rate construction, for each i. This is probably fairly intuitive and we do not go into details here. This so-called, *random time change representation* [21, Chapter 6] is very useful for analysis

since we are able to consider the entire family of processes, $\{\mathbf{N}^i(t)\}_{i=1}^{\infty}$, on the same probability space in terms of the same small number of stochastic primitives, $\{P^k(t)\}_{k=1}^{K}$. It is a key device used in the proof of the functional central limit theorem, which follows.

Theorem 2.1 *Let $T > 0$ and let $\hat{\mathbf{T}}$ be the subset of $\{t \in [0,\, T)\}$ for which $\mathbf{f}(\cdot)$ is not totally differentiable at the point $\mathbf{v}(t)$. We require that $\hat{\mathbf{T}}$ has Lebesgue measure zero. Then on all of $[0,\, T) \setminus \hat{\mathbf{T}}$, $\mathbf{f}(\cdot)$ has a well-defined Jacobian at the point $\mathbf{v}(t)$, say $D\,\mathbf{f}(\mathbf{v}(t))$. Extend this to all points $\{\mathbf{v}(t) \,:\, t \in [0,\, T)\}$, say by defining it to be the matrix of zeros at times in $\hat{\mathbf{T}}$.*

Then if $S_i \to \infty$ as $i \to \infty$, then:

$$\frac{\mathbf{N}^i(t)}{\sqrt{S_i}} - \sqrt{S_i}\mathbf{v}(t) \Rightarrow \mathbf{E}(t)$$

where:

$$\mathbf{E}(t) := \int_0^t D\,\mathbf{f}(\mathbf{v}(s)) \cdot \mathbf{E}(s)\,ds + \sum_{k \in K} W^k\left(\int_0^t f^k(\mathbf{v}(s))\,ds\right)\mathbf{l}^k$$

$\{W^k(t)\}_{k=1}^{K}$ is a sequence of K mutually independent standard Wiener processes (aka Brownian motions) and the convergence is weak convergence in $D_{\mathbb{R}_+^N}[0,\, T)$, the space of $\mathbb{R}_+^N$-valued càdlàg[3] functions, equipped with the Skorohod J_1 topology.

Proof. We do not present the proof in detail in this paper, however, a very brief outline is given in Appendix A.1. □

This theorem suggests the following approximation for $\mathbf{N}^i(t)$:

$$\mathbf{N}^i(t) \approx S_i\mathbf{v}(t) + \sqrt{S_i}\mathbf{E}(t)$$

Figure 2 shows some comparisons of traces of this second-order approximation with traces of the actual underlying CTMC for the client/server model. We see the presence of the expected statistical regularity between the two processes. More useful, however, to validate the results of the above theorem is a comparison of the root-scaled divergence of the fluid approximation from the CTMC:

$$\frac{\mathbf{N}^i(t) - S_i\mathbf{v}(t)}{\sqrt{S_i}}$$

with its approximating process $\mathbf{E}(t)$. Figure 3 shows trace comparisons of these two stochastic processes and we can see that as we increase the component populations, the statistical regularity between the two processes does appear to be increasing.

2.2 SDEs and Fokker-Planck equations

It is possible to show that the stochastic process $\mathbf{E}(t)$ defined above is equal in distribution (on $D_{\mathbb{R}_+^N}[0,\, \infty)$) to the unique solution, $\bar{\mathbf{E}}(t)$ of the following (It$\bar{\text{o}}$) stochastic differential equation (SDE):

$$d\bar{\mathbf{E}}(t) = \mu(\bar{\mathbf{E}}(t),\, t)\,dt + \sigma(t)\,d\mathbf{W}(t)$$

where $\mu(\mathbf{x},\, t) : \mathbb{R}^N \times \mathbb{R}_+ \to \mathbb{R}^N$ and $\sigma(t) : \mathbb{R}_+ \to \mathbb{R}^{N \times K}$ are defined by:

$$\mu(\mathbf{x},\, t) := D\,\mathbf{f}(\mathbf{v}(t)) \cdot \mathbf{x}$$
$$\sigma(t) := \left(l_i^j \times \sqrt{f^j(\mathbf{v}(t))}\right)_{ij}$$

and $\mathbf{W}(t)$ is a K-dimensional standard Wiener process.

[3] Continue à droite, limitée à gauche, that is, right continuous with left limits.

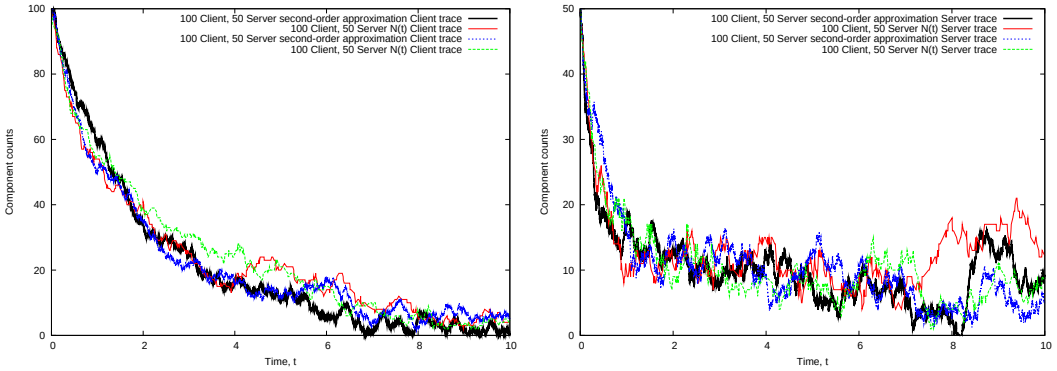


Fig. 2. Comparison of second-order approximation traces with traces of the $Client$ and $Server$ counting processes for the client/server model. Rates used are $r_{req} = 3.0$, $r_{think} = 0.3$, $r_{break} = 0.3$, $r_{data} = 1.0$ and $r_{reset} = 0.2$.

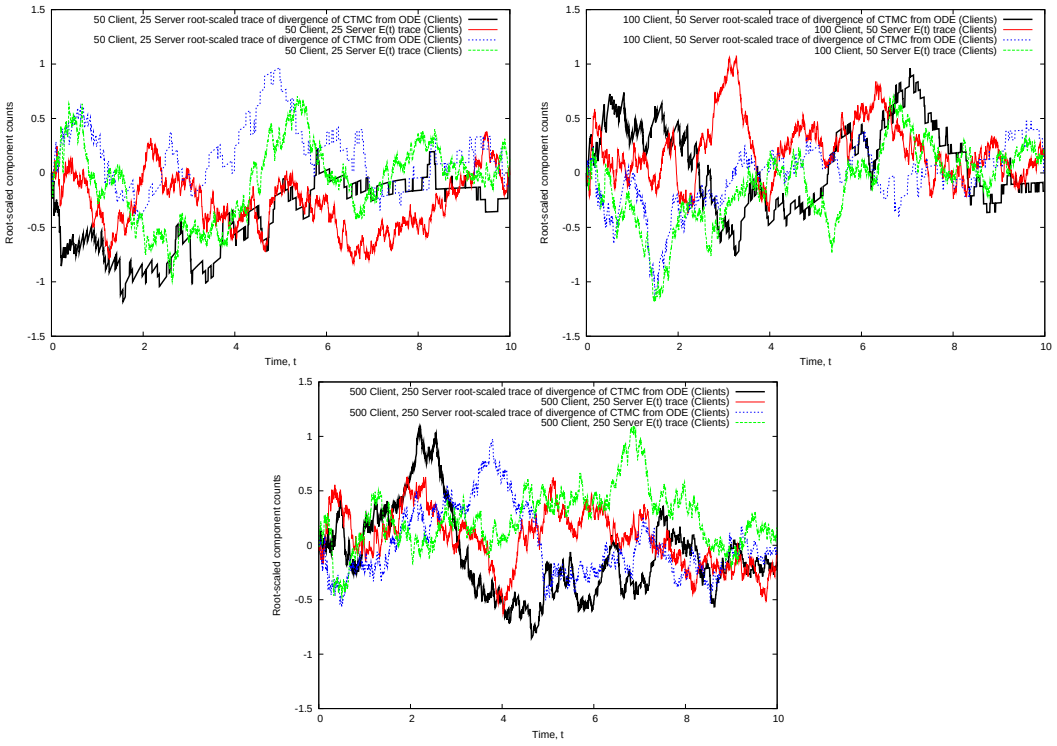


Fig. 3. Comparison of $\mathbf{E}(t)$ traces for $Client$ components with traces of the corresponding root-scaled divergence of the CTMC from the ODE approximation for the client/server model. We show three figures, increasing the total component population each time. Rates used are $r_{req} = 3.0$, $r_{think} = 0.3$, $r_{break} = 0.3$, $r_{data} = 1.0$ and $r_{reset} = 0.2$.

Furthermore, from this, we may derive the Fokker-Planck partial differential equation (PDE) (see e.g. [22]), which governs the evolution of the time-dependent probability density, $p(\mathbf{x},\ t) : \mathbb{R}^N \times \mathbb{R}_+ \to \mathbb{R}_+$ of $\bar{\mathbf{E}}(t)$ (and thus also of $\mathbf{E}(t)$). To state this equation, define first $g_i^1(\mathbf{x},\ t) : \mathbb{R}^N \times \mathbb{R}_+ \to \mathbb{R}$ and $g_{ij}^2(t) : \mathbb{R}_+ \to \mathbb{R}$

by:

$$g_i^1(\mathbf{x},\, t) := \mu_i(\mathbf{x},\, t)$$
$$g_{ij}^2(t) := \frac{1}{2}\sum_{k=1}^{K}\sigma_{ik}(t)\sigma_{jk}(t)$$

then the Fokker-Planck PDE is:

$$\frac{\partial p}{\partial t} = -\sum_{i=1}^{N}\frac{\partial}{\partial x_i}\left[g_i^1(\mathbf{x},\, t)p(\mathbf{x},\, t)\right] + \sum_{i=1}^{N}\sum_{j=1}^{N} g_{ij}^2(t)\frac{\partial^2}{\partial x_i \partial x_j}\left[p(\mathbf{x},\, t)\right]$$

Solving this analytically is possible only in a few special cases, however, numerical integration may be possible where N is not too large. This would provide a route to direct approximation of probability distributions of interest in the original CTMC. Furthermore, since in our case, σ and thus g_{ij}^2 have no dependence on the state vector, we are actually dealing with a special subset of the general class of Fokker-Planck equations, which may yield more easily to numerical solution, or otherwise. It is also highly probable that simpler systems of equations can be extracted from this PDE for specific quantities of interest, such as first passage-times and steady-state distributions, all of which will be investigated in later papers.

3 Conclusion and future work

We have presented a functional central limit theorem which quantifies, as a stochastic process, the difference between a PEPA model's underlying CTMC and its fluid approximation. Our work is based on a generalisation of the theory found in [21] to handle the case of non-smooth rate functions. We have demonstrated the convergence empirically using stochastic simulation and outlined avenues for exploiting it more analytically in future work.

References

[1] J. Hillston, *A Compositional Approach to Performance Modelling*. Cambridge University Press, 1996.

[2] J. Hillston, "Fluid flow approximation of PEPA models," in *Proceedings of the Second International Conference on the Quantitative Evaluation of Systems*, (Torino, Italy), pp. 33–43, IEEE Computer Society Press, Sept. 2005.

[3] J. T. Bradley, S. T. Gilmore, and J. Hillston, "Analysing distributed internet worm attacks using continuous state-space approximation of process algebra models," *Journal of Computer and System Sciences*, vol. 74, pp. 1013–1032, September 2008.

[4] N. Geisweiller, J. Hillston, and M. Stenico, "Relating continuous and discrete PEPA models of signalling pathways," *Theoretical Computer Science*, vol. 404, pp. 97–111, November 2008.

[5] R. Hayden and J. Bradley, "Evaluating fluid semantics for passive stochastic process algebra cooperation," *Performance Evaluation*, 2009. Accepted for publication.

[6] L. Bortolussi and A. Policriti, "Stochastic concurrent constraint programming and differential equations," in *QAPL'07, 5th Workshop on Quantitative Aspects of Programming Languages*, vol. 190 of *Electronic Notes in Theoretical Computer Science*, pp. 27–42, September 2007.

[7] L. Cardelli, "From processes to ODEs by Chemistry," in *TCS 2008, Fifth IFIP International Conference on Theoretical Computer Science*, (Milan), Springer, 2008.

[8] J. Júlvez, E. Jiménez, L. Recalde, and M. Silva, "On observability in timed continuous petri net systems," in *QEST'04, 1st International Conference on Quantitative Evaluation of Systems*, vol. 266, pp. 60–69, IEEE, September 2004.

[9] A. Duguid, "Coping with the parallelism of BitTorrent: Conversion of PEPA to ODEs in dealing with state space explosion," in *Formal Modeling and Analysis of Timed Systems, 4th International Conference, FORMATS 2006, Paris, France, September 25-27, 2006, Proceedings* (E. Asarin and P. Bouyer, eds.), vol. 4202 of *Lecture Notes in Computer Science*, pp. 156–170, Springer, 2006.

[10] S. Gilmore and M. Tribastone, "Evaluating the scalability of a web service-based distributed e-learning and course management system," in *Third International Workshop on Web Services and Formal Methods (WS-FM'06)* (M. Bravetti, M. T. Núñez, and G. Zavattaro, eds.), vol. 4184 of *Lecture Notes in Computer Science*, (Vienna, Austria), pp. 156–170, Springer, 2006.

[11] M. Bravetti, S. Gilmore, C. Guidi, and M. Tribastone, "Replicating web services for scalability," in *Proceedings of the Third International Conference on Trustworthy Global Computing (TGC'07)* (G. Barthe and C. Fournet, eds.), vol. 4912 of *LNCS*, pp. 222204–221, Springer-Verlag, 2008.

[12] J. Hillston, "Process algebras for quantitative analysis," in *Proceedings of the 20th Annual IEEE Symposium on Logic in Computer Science (LICS' 05)*, (Chicago), pp. 239–248, IEEE Computer Society Press, June 2005.

[13] H. Bowman, J. Bryans, and J. Derrick, "Analysis of a multimedia stream using stochastic process algebra," in *Sixth International Workshop on Process Algebras and Performance Modelling* (C. Priami, ed.), (Nice), pp. 51–69, September 1998.

[14] J. Forneau, L. Kloul, and F. Valois, "Performance modelling of hierarchical cellular networks using PEPA," *Performance Evaluation*, vol. 50, pp. 83–99, Nov. 2002.

[15] N. Thomas, J. T. Bradley, and W. J. Knottenbelt, "Stochastic analysis of scheduling strategies in a GRID-based resource model," *IEE Software Engineering*, vol. 151, pp. 232–239, September 2004.

[16] D. R. W. Holton, "A PEPA specification of an industrial production cell," in *Process Algebra and Performance Modelling Workshop* (S. Gilmore and J. Hillston, eds.), vol. 38(7) of *Special Issue: The Computer Journal*, pp. 542–551, CEPIS, Edinburgh, June 1995.

[17] J. Bradley, N. Dingle, S. Gilmore, and W. Knottenbelt, "Derivation of passage-time densities in PEPA models using IPC: The Imperial PEPA Compiler," in *Proceedings of the 11th IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunications Systems* (G. Kotsis, ed.), (University of Central Florida), pp. 344–351, IEEE Computer Society Press, Oct. 2003.

[18] R. A. Hayden and J. T. Bradley, "Fluid semantics for passive stochastic process algebra cooperation," in *VALUETOOLS'08, Third International Conference on Performance Evaluation Methodologies and Tools*, (Athens), 2008.

[19] T. Kurtz, "Solutions of ordinary differential equations as limits of pure jump Markov processes," *Applied Probability*, vol. 7, pp. 49–58, April 1970.

[20] T. Kurtz, "Strong approximation theorems for density dependent Markov chains," *Stochastic Processes and Applications*, vol. 6, pp. 223–240, 1978.

[21] T. Kurtz and S. Ethier, *Markov Processes Characterisation and Convergence*. Wiley, 1986.

[22] C. Gardiner, *Handbook of Stochastic Methods for Physics, Chemistry, and the Natural Sciences*. Springer, 1983.

A Proofs

A.1 Proof of Theorem 2.1

We work with the random time change representation of $\mathbf{N}^i(t)$:

$$\mathbf{N}^i(t) = \mathbf{N}^i(0) + \sum_{k\in K} P^k\left(\int_0^t f^k(\mathbf{N}^i(s))\,ds\right)\mathbf{l}^k$$

By Corollary 5.5 and Remark 5.4 of [21, Chapter 7], we may state the following lemma:

Lemma A.1 *A standard (rate one) Poisson process, $P(t)$, can be constructed on the same probability space as a standard Wiener process, $W(t)$, such that the random variable:*

$$Z := \sup_{t\in\mathbb{R}_+}\frac{|P(t)-t-W(t)|}{\log(2\vee t)} < \infty \quad \textit{almost surely}$$

Using this lemma, we may construct our CTMC processes, $\{\mathbf{N}^i(t)\}_{i=1}^\infty$ on a probability space equipped with the required K mutually independent standard Poisson processes, $\{P^k(t)\}_{k=1}^K$, but also with K mutually independent standard Wiener processes, $\{W^k(t)\}_{k=1}^K$, such that, we may also define the random variables, $\{Z^k\}_{k=1}^K$:

$$Z^k := \sup_{t\in\mathbb{R}_+}\frac{|P^k(t)-t-W^k(t)|}{\log(2\vee t)} < \infty \quad \text{almost surely}$$

Moreover, define for each $1 \le i < \infty$, $W^{k,\,i}(t) := \frac{1}{\sqrt{S_i}}W^k(S_i t)$, and note that each $W^{k,\,i}(t)$ is also a standard Wiener process by self-similarity. Then we may write:

$$Z^k = \sup_{t\in\mathbb{R}_+}\frac{|P^k(t)-t-\sqrt{S_i}W^{k,\,i}(t/S_i)|}{\log(2\vee t)}$$

for all $1 \le i < \infty$. This allows us to derive the following strong approximation result:

$$\mathbf{N}^i(t) = \mathbf{N}^i(0) + \int_0^t \mathbf{f}\left(\frac{1}{S_i}\mathbf{N}^i(s)\right)\,ds + \sum_{k\in K}\sqrt{S_i}W^{k,\,i}\left(\frac{1}{S_i}\int_0^t f^k\left(\frac{1}{S_i}\mathbf{N}^i(s)\right)\,ds\right)\mathbf{l}^k + O(\log(S_i))$$

almost surely. A direct comparison of $\frac{\mathbf{N}^i(t)-S_i\mathbf{v}(t)}{\sqrt{S_i}}$ with $\mathbf{E}(t)$ as defined in the statement of the theorem, using this strong approximation then yields the result. We omit further details here. □

Qualitative Reasoning of Stochastic Models and the Role of Flux

Paolo Ballarini[*], Maria Luisa Guerriero[†], and Jane Hillston[‡]

Abstract

Qualitative reasoning (QR) is a technique integrating the fields of AI and systems theory, whose aim is to be able to reason about the behaviour of systems with uncertainty in parameter values and in the exact quantitative dynamics. Traditionally applied to the study of the dynamics of physical systems, QR has been usually considered in a deterministic setting. Here we investigate the application of a QR approach to the analysis of continuous time Markov chains (CTMCs), and we focus on the application of probabilistic fluxes analysis as a first experiment in this context.

1 Introduction

Qualitative reasoning (QR) is an approach to analysing dynamic systems at a level of abstraction which disregards precise quantitative parameters but nevertheless is able to derive information about the possible behaviours which might be exhibited by the system [6, 7]. It has been primarily studied in AI and can be regarded as a form of knowledge representation. It is sometimes likened to *common-sense* reasoning, the analogy being that we can all predict the behaviour of many physical dynamic systems, such as the swinging of a pendulum, without having to analyse the differential equation which precisely governs that behaviour. Motivations for QR include uncertainty about parameter values, uncertainty about the initial conditions of the system, or even explicit uncertainties in the model, for example with stochastic differential equations [4]. Applications of QR include diagnostics, fault prediction, planning, argumentation and explanation.

Recently QR has been proposed in the context of systems biology [1, 5]. In biochemical systems biologists often characterise behaviour based on the qualitative trends of proteins concentrations rather than quantitative amounts. There are several reasons for this:

- parameter values for models of biochemical systems are rarely known in detail;
- the models themselves may be based on partial knowledge and speculation;
- difficulties in exactly repeating the same experiment, and the indirect forms of measurement used, make it difficult to characterise a behaviour quantitatively.

Given its origins in modelling physical systems QR is usually applied to systems of differential equations which have deterministic behaviour. The motivation for this paper is to consider whether there is scope to develop an approach to QR for stochastic models of dynamic behaviour.

[*]CoSBi, Trento, Italy. Email: ballarini@cosbi.eu
[†]University of Edinburgh, Scotland, UK. Email: mguerrie@inf.ed.ac.uk
[‡]University of Edinburgh, Scotland, UK. Email: jane.hillston@ed.ac.uk

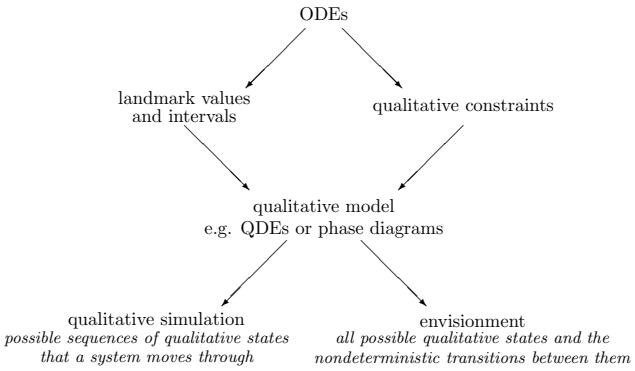


Figure 1: Schematic view of QR reasoning.

This is preliminary work and the current paper is intended to explore the issues which may be involved rather than present any definitive answers.

2 Qualitative Reasoning

A number of different styles of qualitative modelling have appeared in the literature but they all start from a continuous, generally deterministic, representation of the dynamics of a system, usually as a set of ordinary differential equations (ODEs). Abstraction is applied to the system so that rather than look at the continuum of possible values for variables within the equations, instead a discrete set of intervals is identified. The boundaries between intervals are important and termed *landmark* values for the variables. In the simplest case the landmark values might be just $-\top, 0, \top$, creating the intervals $(-\top, 0)$ and $(0, \top)$ i.e. we only distinguish whether a value is negative or positive. If the differential equation governing the behaviour of the variable includes higher derivatives (i.e. the differential equation is higher than first order) each derivative is treated as a distinct variable. Thus if we consider a ball thrown vertically into the air, its height at time t is given by

$$x(t) = -\frac{1}{2}gt^2 + v_0 t + x_0$$

and the variables correspond to the height and the velocity. The height may be positive or 0, whereas the velocity may be positive, negative or zero.

Which ever style of qualitative modelling is used, the steps of QR are broadly the same and as illustrated in Figure 1. The potential qualitative states can be found by considering the product space derived from the qualitative states of each variable. However, in general it will not be possible to reach all these states as the dynamics of the system places constraints on its behaviour. For example, once the velocity of the ball has reached zero it is no longer possible for the height of the ball to increase. Applying the qualitative constraints to the product state space reduces it to the qualitative model which may be represented as a phase diagram or a set of qualitative differential equations for example. In essence this captures which combinations of intervals and values for variables are valid. This representation may then be subjected to *simulation* to derive valid trajectories, or *envisionment*. An envisionment is the state transition diagram for the qualitative state space. It may be either a *total* envisionment, considering all valid states, or an *attainable* envisionment, showing only those states reachable from known

initial conditions.

In some systems we may gain more understanding of the behaviour of the system if as well as the qualitative value of state variables we also have qualitative information about its derivative. For example, if we consider a swinging pendulum, knowing that the velocity is positive does not characterise the different behaviours in different parts of the trajectory. However if we also know how the rate of change of velocity is varying we gain a more complete view of the behaviour. Thus some QR systems incorporate not just variable intervals and landmarks into their representation of qualitative states, but also derivative intervals and landmarks [2].

There has been some work interpreting the envisionment as the state transition diagram for a Markov process and then subjecting it to usual Markov analysis [4]. In this case probabilities must be attributed to each of the transitions within the state space. In [4] this is done using a relative frequency argument. Doyle and Sacks work with a phase transition QR model and choose the probabilities according to the proportion of points within the volume of phase space corresponding to the abstract state which will follow a trajectory leading to another abstract state within one time unit. Once the Markov chain is generated they propose analysing it to identify transient and ergodic states, as well as usual Markovian analysis to find steady state and time to absorption in the case of an absorbing Markov chain. Note that this is quite different from what we are interested in. Here the stochasticity is added only after the derivation of the qualitative state space as a means to assess which (deterministic) behaviour is more likely. We are interested in replacing the dynamic model on which the QR is based by a continuous time Markov chain (CTMC).

3 Flux-based Analysis: a First Experiment on Stochastic Qualitative Reasoning

We will consider CTMC models which might arise from consideration of systems composed of interacting components, for example PEPA models. Thus the dimensions of our CTMC will correspond to the possible local states of the components, and our state representation will be the count of the number of components exhibiting each local state. These will be variables which QR would seek to abstract. Therefore note that the high/low style of PEPA modelling, and Bio-PEPA with levels, could already be viewed as qualitative representations of the system, with the PEPA/Bio-PEPA semantics providing the qualitative constraints, and the derived state space being an attainable envisionment.

However this is not the focus of our current experiment. Here we aim to investigate alternative, possibly more abstract, ways of viewing the dynamics of a CTMC. In general, the quantitative dynamics of CTMC are studied in two ways.

- By simulation, in which one possible trajectory through the state space is generated at a time, exhibiting one possible result of the stochastic choices within the system. Multiple trajectories must be generated before any conclusions can be drawn about the behaviour of the system. However the behaviour which is observed is definitive in the sense that there is a clear progression from state to state.
- By numerical solution of the state probability distribution, considering all possible behaviours at the same time. Here what is observed is not a progression from state to state but a shifting of probability mass within the probability distribution as some states become more likely than others over time.

It is not straightforward to map either of these directly to the ODE characterisation of system dynamics on which QR is usually based. Like simulation the ODE could be regarded as a linear

time view, as a single trajectory is observed. However, in many ways this trajectory is related to the expected value of state variables which might be observed in numerical solution of the CTMC as the ODE dynamics allows all possible actions to occur at once. In other words the ODE trajectory corresponds to the expected value of the state variables in the CTMC. Thus we have chosen this latter view of the CTMC and wish to consider how analysis of the probability distribution and the forces governing its evolution may be used for QR.

CTMC models are characterised by a *steady-state* and a *transient window*. Starting at time $t = 0$ the model eventually reaches its steady-state within a given time $t_{ss} > 0$. The time interval $[0, t_{ss}]$ is the *transient window* of the CTMC model. However note that this notion of steady-state is quite different to that considered in the QR literature, where due to the deterministic nature of the dynamic model the steady-state is a single, absorbing state.

In the rest of this section we focus on *probabilistic fluxes* into/out of a given state of a (finite state) CTMC model, and we investigate their use for the analysis of the (qualitative) dynamic behaviour of the CMTC. First we give a few relevant definitions, and then we show the result of flux-based analysis on a few simple systems.

An $(n \geq 1)$-dimensional CTMC M is characterised by a set of n non-negative integer valued variables $(X^1, X^2, \cdots, X^n)$, and a state $s_j = (x_j^1, x_j^2, \cdots, x_j^n) \in S$ is an n-tuple representing the values of each variable X^i in state s_j.

A transition $q : s_j \to s_k$ (with $s_j = (x_j^1, x_j^2, \cdots, x_j^n)$ and $s_k = (x_k^1, x_k^2, \cdots, x_k^n)$) such that $Q(s_j, s_k) > 0$ is called an X^i-*increasing* transition if and only if $(x_j^i < x_k^i)$. Similarly, q is called X^i-*decreasing* iff $(x_j^i > x_k^i)$, and X^i-*invariant* iff $(x_j^i = x_k^i)$.

Given an n-dimensional CTMC $M = (S, Q, s_0)$, a state $s_j = (x_j^1, x_j^2, \cdots, x_j^n) \in S$, and a variable X^i (with $1 \leq i \leq n$), we define $E_{+i}(s_j), E_{-i}(s_j), E_{=i}(s_j) \leq E(s_j)$ as the outgoing rates from state s_j corresponding to, respectively, X^i-increasing, X^i-decreasing and X^i-invariant transitions (and we call them X^i-increasing, X^i-decreasing, and X^i-invariant outgoing rates). Formally, they are defined in Eq. (1), Eq. (2) and Eq. (3), respectively.

$$E_{+i}(s_j) = \begin{cases} \sum_{1 \leq k \leq m, k \neq j} Q(s_j, s_k) & \textbf{if } \; x_j^i < x_k^i \\ 0 & \textbf{otherwise} \end{cases} \tag{1}$$

$$E_{-i}(s_j) = \begin{cases} \sum_{1 \leq k \leq m, k \neq j} Q(s_j, s_k) & \textbf{if } \; x_j^i > x_k^i \\ 0 & \textbf{otherwise} \end{cases} \tag{2}$$

$$E_{=i}(s_j) = \begin{cases} \sum_{1 \leq k \leq m, k \neq j} Q(s_j, s_k) & \textbf{if } \; x_j^i = x_k^i \\ 0 & \textbf{otherwise} \end{cases} \tag{3}$$

These tell us the rates of change to particular variables in particular states but since we view the system through its probability distribution we need to instead consider the expected values for these rates of change, i.e. the probabilistic flux. Given a state $s \in S$, and a time instant $t \in \mathbb{R}_+$, we define the *instantaneous state out-flux* as follows.

Definition 1 (Instantaneous state out-flux) *The* instantaneous state out-flux *for s at time t is defined as:*

$$flux(s, t) = \pi(s, t) \cdot E(s) = \pi(s, t) \cdot -Q(s, s) \ .$$

The out-flux for state s at time t is computed as the product of the transient probability of being in state s at time t (i.e. $\pi(s, t)$) multiplied by the emanating rate from s (i.e. the sum of rates of all its outgoing transitions).

We can now define the i^{th}-*dimensional instantaneous state out-flux* and the i^{th}-*dimensional instantaneous total out-flux*.

Definition 2 (Instantaneous i^{th}-dimensional state out-flux) *The* instantaneous i^{th}-dimensional increasing/descreasing/invariant state out-fluxes *for s at time t (with $1 \leq i \leq n$) are defined as:*

$$flux_{+_i}(s,t) = \pi(s,t) \cdot E_{+i}(s)$$
$$flux_{-_i}(s,t) = \pi(s,t) \cdot E_{-i}(s)$$
$$flux_{=_i}(s,t) = \pi(s,t) \cdot E_{=i}(s)$$

where $E_{+i}(s), E_{-i}(s), E_{=i}(s) \leq E(s)$ are the X^i-increasing, X^i-decreasing, and X^i-invariant outgoing rates from s.

The i^{th}-dimensional increasing/decreasing/invariant state out-flux are, hence, the portions of $flux(s,t)$ corresponding to, respectively, X^i-*increasing*, X^i-*decreasing* and X^i-*invariant* transitions.

The X^i-*instantaneous gradient state out-flux*, denoted as $flux_{\delta i}(s,t)$ is defined as the difference between the increasing and the decreasing fluxes:

$$flux_{\delta i}(s,t) = flux_{+_i}(s,t) - flux_{-_i}(s,t) \ .$$

Definition 3 (Instantaneous i^{th}-dimensional total out-flux) *The* instantaneous i^{th}-dimensional increasing/descreasing/invariant/gradient total out-fluxes *at time t (with $1 \leq i \leq n$) are defined as:*

$$flux_{+i}(t) = \sum_{s \in S} flux_{+i}(s,t)$$
$$flux_{-i}(t) = \sum_{s \in S} flux_{-i}(s,t)$$
$$flux_{=i}(t) = \sum_{s \in S} flux_{=i}(s,t)$$
$$flux_{\delta i}(t) = \sum_{s \in S} flux_{\delta i}(s,t)$$

The total i^{th}-dimensional out-fluxes (increasing/decreasing/invariant/gradient) for the CTMC M at time t are the sum, over all states of M, of the i^{th}-dimensional increasing/decreasing and invariant/gradient fluxes at time t (again, for the sake of readability, M is omitted from the notation of total fluxes).

Definition 4 (i^{th}-dimensional flux analysis of CTMC) *Let $T = [t_1, t_2]$ be a time interval (with $t_1 < t_2 \in \mathbb{R}^+$), $F \in \mathbb{N}^*$ be the sampling frequency of T, and $\Delta = \| \frac{(t_2 - t_1)}{F} \|$ be the sampling period denoting the amplitude of each sub-interval of T. The i^{th}-**dimensional flux analysis** for a given CTMC M over the time interval T is achieved through calculation of the increasing/decreasing/invariant/gradient fluxes for M over the time interval T and with respect to the sampling period Δ.*

In the next section we show some results of the flux-analysis for a few simple models.

3.1 Flux analysis of CTMC: Some Examples

We consider few examples of multi-dimensional CTMCs modelling elementary biochemical systems. The number of dimensions of each such a CTMC corresponds to the number of biochemical species in the considered system. In each one of the following examples we calculate the positive, the negative, the invariant as well as the gradient flux of each dimension (i.e. species). We discuss the obtained results by comparing the flux behaviour with classical transient analysis.

3.1.1 $A \to B \to C$

We consider the 3-dimensional CTMC model corresponding to the biochemical system described by reactions:

$$\begin{aligned} A &\to B \\ B &\to C \end{aligned} \tag{4}$$

Equations (4) are an example of simple *closed system* (i.e. the total population is invariant) representing the transformation of molecules of a source species A into molecules of a target species C via an intermediate species B. For simplicity we assume the two reactions to be *balanced* (i.e. having the same rate) and we consider an initial configuration with only 10 molecules of A (i.e. we assume $(10, 0, 0)$ to be the initial state). We observe that the corresponding CMTC has $(n+1)(n+2)/2$ states (where $n = a_0$ is the initial amount of A) with exactly one deadlock state (i.e. $(0, 0, n)$). The dynamics of such a system is intuitively pretty simple: starting at time $t = 0$ species A will monotonically decrease to 0, while its dual, species C, will monotonically increase to $n = 10$. On the other hand the intermediate species B will have a peak corresponding to the progressive *transformation* of A into C. The *transient* behaviour of species A, B and C is depicted in Figure 2, where each line represents the *instant reward* of a version of the CTMC model with state rewards (i.e. rewards corresponding to the level of each species have been assigned to each state of the chain): this corresponds to the expected value of the species at that time. Figure 2 reflects the intuition with A and C monotonically increasing and decreasing, respectively, throughout the transient window, and with B peaking after about 1 time unit.

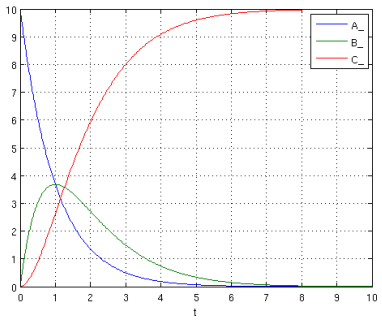
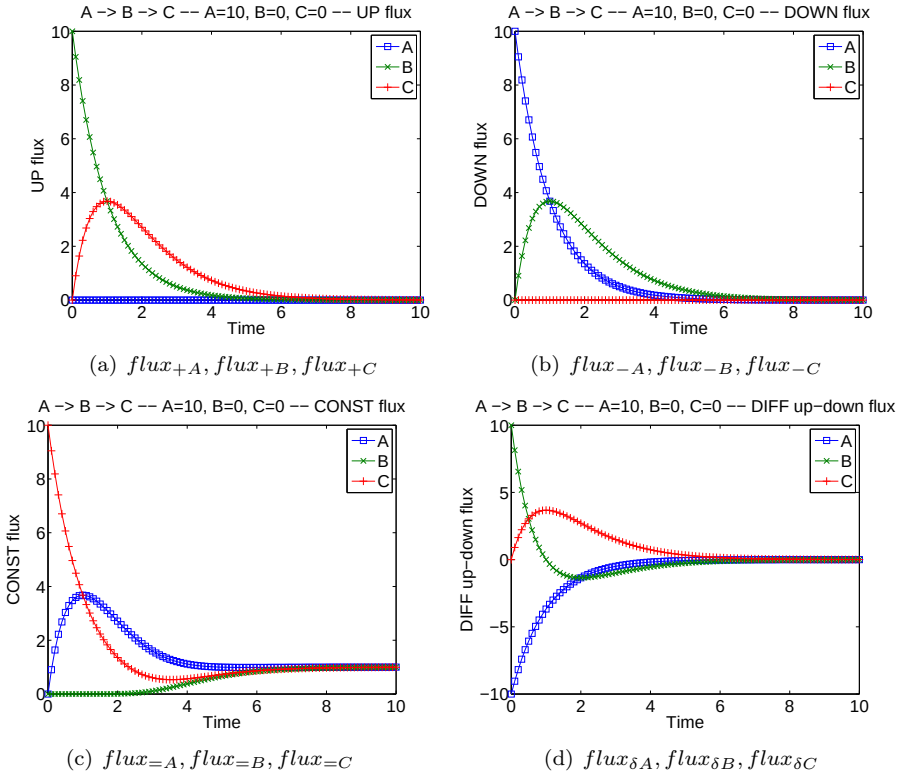


Figure 2: Instant reward (transient analysis) of the $A \to B \to C$ model.

The interpretation of fluxes is also in compliance with the intuitive description of the system behaviour, however it requires a little bit more thinking. With respect to the positive fluxes (Figure 3(a)) we observe that A-positive flux (blue line) is constantly null, as A cannot increase at all in this system. On the other hand, B-positive flux (green line), starting from its maximum level (corresponding to the maximal rate of conversion of A into B), steadily decreases up until

all A molecules have turned into B molecules. Finally the C-positive flux for (red line) shows a peak representing the maximal speed at which C is produced and which happens slightly after B has reached its peak (i.e. at $t = 1$, compare the green line of Figure 2 with the red line of Figure 3(a)), which is: when half of the A molecules has turned into B molecules. The interpretation of the negative fluxes (Figure 3(b)) and invariant fluxes (Figure 3(c)) can be devised similarly. Finally we spend a few words commenting on the gradient fluxes (Figure 3(d)) because they provide the most significant source of *qualitative* information about the dynamics of a CTMC model. Figure 3(d) illustrates the *qualitative* behaviour of the system within the transient window. Specifically we observe that A-gradient flux (blue line) is always negative, signifying that the overall amount of A will constantly decrease. On the other hand, C-gradient flux (red line) is always positive, meaning that the overall amount of C will only increase in the system, although the presence of a peak tells us that the speed of such growth will progressively slow down after an initial *acceleration*. Finally, B-gradient flux (green line) spans from (strongly) positive to (weakly) negative, meaning that after an initial fast growth, the increase in B will slow down and eventually turn into a decrease.



(a) $flux_{+A}, flux_{+B}, flux_{+C}$ (b) $flux_{-A}, flux_{-B}, flux_{-C}$

(c) $flux_{=A}, flux_{=B}, flux_{=C}$ (d) $flux_{\delta A}, flux_{\delta B}, flux_{\delta C}$

Figure 3: Fluxes calculation for the $A \rightarrow B \rightarrow C$ model.

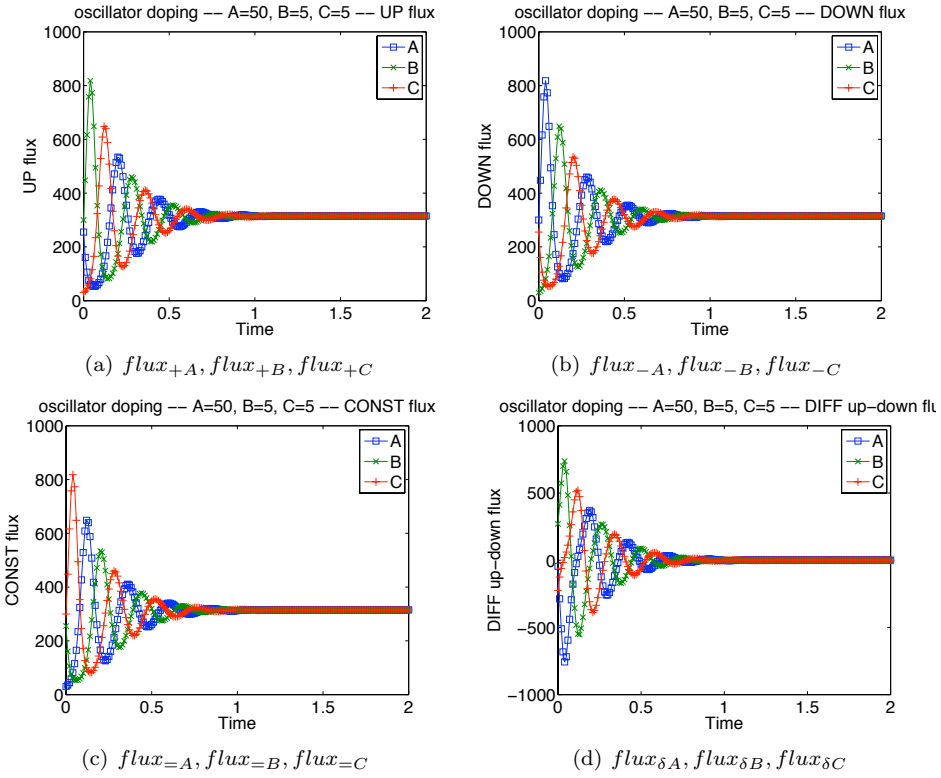
3.1.2 A 3-way oscillator.

As a further example of CTMC flux analysis we consider a slightly more complicated system, known as the 3-way oscillator [3]. The model is (as in the previous case) a 3-dimensional CTMC

model corresponding to the biochemical system described by the following 3 + 3 reactions[1]:

basic reactions	**doping reactions**
$A + B \rightarrow B + B$	$D_A + B \rightarrow A + D_A$
$B + C \rightarrow C + C$	$D_B + C \rightarrow B + D_B$
$C + A \rightarrow A + A$	$D_C + A \rightarrow C + D_C$

Species D_A, D_B and D_C represent *doping substances* for the main species, respectively A, B and C. Stochastic simulations show a permanent oscillation for the above system: the levels of molecules for the three species A, B and C fluctuate permanently and in a co-ordinated fashion. The presence of the *doping reactions* guarantees that the oscillation does never stop. In this case the resulting CTMC model is finite-state and ergodic (no deadlock states). We also consider the *no-doping* variant of such model, whereby doping reactions (and species) are removed. In this case the system still oscillates; however, the oscillation damps down in finite time (i.e. the underlying CTMC has 3 deadlock states corresponding to all molecules accumulating in either A, B or C).



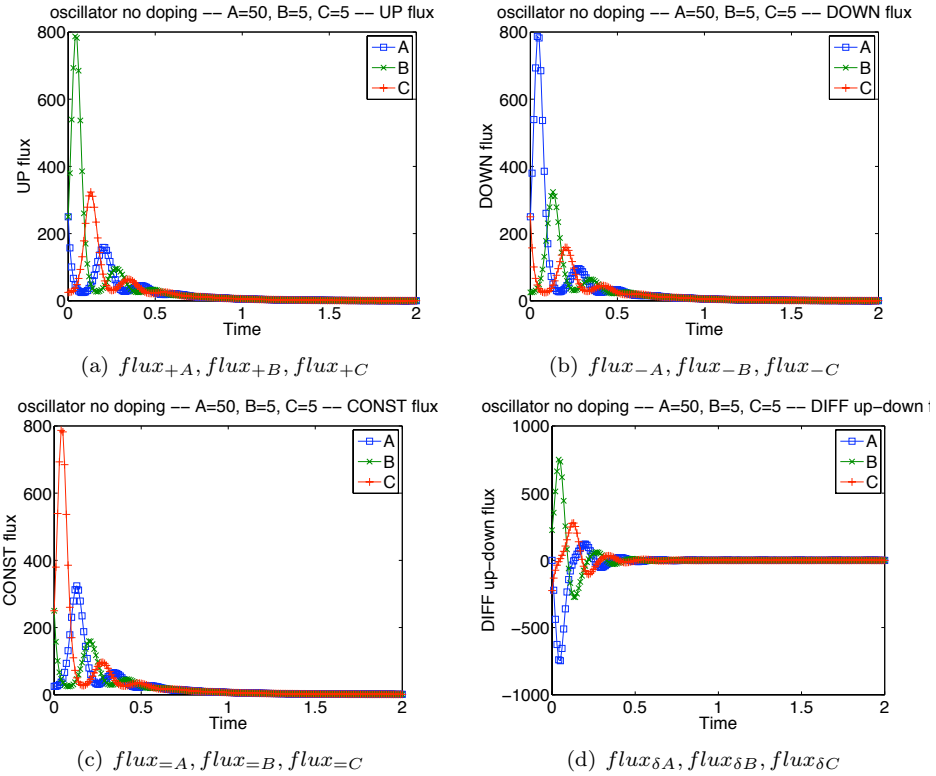
(a) $flux_{+A}, flux_{+B}, flux_{+C}$ (b) $flux_{-A}, flux_{-B}, flux_{-C}$

(c) $flux_{=A}, flux_{=B}, flux_{=C}$ (d) $flux_{\delta A}, flux_{\delta B}, flux_{\delta C}$

Figure 4: Fluxes calculation for the 3-way oscillator model with doping.

[1]Note that even though there are actually 6 species in this systems, the underlying CTMC is still 3-dimensional, as three of the species, namely D_A, D_B and D_C, are just *modifiers* (i.e. neither *reactants* nor *products*), and thus their amounts do not need to be captured in the state.

Results of flux calculations for a version of the model with initial state $(50, 5, 5)$ are showed in Figure 4 for the oscillator with doping (i.e. sustained oscillation) and in Figure 5 for the oscillator without doping reactions (i.e. damped oscillation).



(a) $flux_{+A}, flux_{+B}, flux_{+C}$

(b) $flux_{-A}, flux_{-B}, flux_{-C}$

(c) $flux_{=A}, flux_{=B}, flux_{=C}$

(d) $flux_{\delta A}, flux_{\delta B}, flux_{\delta C}$

Figure 5: Fluxes calculation for the 3-way oscillator model without doping.

In both cases the oscillatory behaviour of the system is reflected by the fluxes, although the sustainability of the doped oscillator clearly cannot be observed through fluxes because of their *transient nature*: the oscillatory trend of the fluxes fades away as the steady state is approached.

The main difference between the doped and non-doped oscillators is reflected by the fact that at steady-state fluxes stabilise at a positive value for the sustained oscillator, and at zero for the damped oscillator. Furthermore, if we compare, for example, the positive fluxes of the sustained (Figure 4(a)) and damped (Figure 5(a)) oscillators, we observe that C-positive flux grows much faster in the doped oscillator than in the non-doped one. This is due to the presence of the doping reactions (and, specifically, of reaction $D_C + A \rightarrow C + D_C$) which boost the production of C molecules proportionally to the amount of A (which in the initial state $(50, 5, 5)$ is indeed large); in the non-doped case, instead, the growth of C is due, exclusively, to transformation of B into C (i.e. reaction $B + C \rightarrow C + C$) which, due to the low (initial) level of B, makes the growth of C much slower.

4 Conclusion

In this paper we have explored the information which can be gained from considering the probabilistic flux acting on state variables in a CTMC. We have demonstrated that this provides an interesting alternative view of the dynamics of the system. However this cannot currently be regarded as qualitative reasoning as both states and rates are represented in full and not abstracted. Moreover, the approach we take, by sampling the probabilistic flux using transient analysis of the CTMC, is computationally intensive and would not be feasible for large systems. Nevertheless this appears to be a rich topic for further exploration.

References

[1] G. Batt, D. Ropers, H. de Jong, J. Geiselmann, R. Mateescu, M. Page, and D. Schneider. Analysis and verification of qualitative models of genetic regulatory networks: A model-checking approach. In L.P. Kaelbling and A. Saffiotti, editors, *Nineteenth International Joint Conference on Artificial Intelligence (IJCAI'05)*, pages 370–375, 2005.

[2] A.M. Bruce and G.M. Coghill. The JMORVEN fuzzy qualitative reasoning software package. `http://www.abdn.ac.uk/~csc282/JMorven/`, 2009.

[3] Luca Cardelli. Artificial biochemistry. Technical report, The Microsoft Research - CoSBi, 2006.

[4] J. Doyle and E.P. Sacks. Markov Analysis of Qualitative Dynamics. *Computational Intelligence*, 7(1):1–10, 1991.

[5] Ross D. King, Simon M. Garrett, and George M. Coghill. On the use of qualitative reasoning to simulate and identify metabolic pathways. *Bioinformatics*, 21(9):2017–2026, 2005.

[6] B. Kuipers. *Qualitative Reasoning: Modeling and simulation with incomplete knowledge*. MIT Press, 1994.

[7] H. Werthner. *Qualitative Reasoning: Modeling and the generation of behavior*. Springer-Verlag, 1994.

A New Deadlock-Checking Method for PEPA

Jie Ding * Jane Hillston †

August 18, 2009

Abstract

This paper presents the Place/Transition structure underlying PEPA models. Based on this structure and techniques developed for Petri nets, an efficient deadlock-checking method is proposed, which avoids the state-space explosion problem.

1 Introduction

Deadlock-checking is an important topic in qualitative analysis of computer and communication systems. The current deadlock-checking algorithm for PEPA relies on exploring the entire state space to find whether a deadlock exists. For large scale PEPA models, deadlock-checking becomes impossible due to the state-space explosion problem. In this paper we will show that there is a Place/Transition (P/T) structure underlying each PEPA model. Based on the techniques developed in the context of P/T systems in Petri nets, we will present a structure-based deadlock-checking approach for PEPA which avoids the state-space explosion problem.

The PEPA models we consider satisfy two assumptions: there is no cooperation within groups of components of the same type; and each column of the activity matrix of a model is distinct, i.e. each labelled activity is distinct in terms of pre and post local derivatives (this will be explained in detail later).

2 Place/Transtion Structure underlying PEPA Models

In [5] Hillston defined a numerical state vector to numerically and compact represent the states of PEPA models. This approach was expanded in [3], where the labelled activity and activity matrices were defined to capture structural information of a model. Based on these definitions, this section will establish that there is a P/T structure underlying each PEPA model. We begin by recalling the activity matrices definitions and the concept of a P/T system.

2.1 Background definitions

In the modified definition of activity matrix, the *labelled activity matrix*, we take care to distinguish activities which have different outcomes even if they involve the same components. To aid in doing this we split the components associated with an activity into *pre* and *post* sets. When we are not concerned with the rate we will denote a transition between local derivatives U and V by $U \xrightarrow{l} V$; to take into account the different rates which may be exhibited by an action in different contexts we may write $U \xrightarrow{(l,r_l^{U\to V})} V$.

*{LFCS, School of Informatics; IDCom, School of Engineering}, University of Edinburgh. Email address: j.ding@ed.ac.uk.

†LFCS, School of Informatics, University of Edinburgh. Email address: jane.hillston@ed.ac.uk.

Definition 1. ***(Pre and post local derivatives)***

1. *If a local derivative U can enable an activity l, i.e. $U \xrightarrow{l} \cdot$, then U is called a* pre local derivative *of l. The set of all pre local derivatives of l is denoted $pre(l)$ and called the* pre set *of l.*

2. *If V is a local derivative obtained by firing an activity l, i.e. $\cdot \xrightarrow{l} V$, then V is called a* post local derivative *of l. The set of all post local derivatives is denoted $post(l)$ and called the* post set *of l.*

3. *The set of all local derivatives derived from U by firing l, i.e. $post(U, l) = \{V \mid U \xrightarrow{l} V\}$, is called the* post set of l from U.

In order to distinguish activities with different outcomes we define the *labelled activity*:

Definition 2. ***(Labelled Activity)***.

1. *For any individual activity l, for each $U \in pre(l), V \in post(U, l)$, label l as $l^{U \to V}$.*

2. *For a shared activity l, for each*

$$(V_1, V_2, \cdots, V_k) \in post(pre(l)[1], l) \times post(pre(l)[2], l) \times \cdots \times post(pre(l)[k], l),$$

label l as l^w, where

$$w = (pre(l)[1] \to V_1, pre(l)[2] \to V_2, \cdots, pre(l)[k] \to V_k).$$

Each $l^{U \to V}$ or l^w is called a ***labelled activity****. The set of all labelled activities is denoted by $\mathcal{A}_{label}$. For labelled activities $l^{U \to V}$ and l^w, their respective pre and post sets are defined as*

$$pre(l^{U \to V}) = \{U\}, \quad post(l^{U \to V}) = \{V\}, \quad pre(l^w) = pre(l), \quad post(l^w) = \{V_1, V_2, \cdots, V_k\}.$$

According to Definition 2, each $l^{U \to V}$ or l^w has a unique output. Note that no new activities are created, since labels are only attached to an existing activity to distinguish the results after this activity being fired.

The impact of labelled activities on local derivatives can be recorded in a matrix form, as defined below.

Definition 3. ***(Activity Matrix, Pre Activity Matrix, Post Activity Matrix)***. *For a model with $N_{\mathcal{A}_{label}}$ labelled activities and $N_{\mathcal{D}}$ distinct local derivatives, the activity matrix $\mathbf{C}$ is an $N_{\mathcal{D}} \times N_{\mathcal{A}_{label}}$ matrix, and the entries are defined as following*

$$\mathbf{C}(U_i, l_j) = \begin{cases} +1 & \text{if } U_i \in post(l_j) \\ -1 & \text{if } U_i \in pre(l_j) \\ 0 & \text{otherwise} \end{cases}$$

where l_j is a labelled activity. The pre activity matrix $\mathbf{C^{pre}}$ and post activity matrix $\mathbf{C^{post}}$ are defined as

$$\mathbf{C^{Pre}}(U_i, l_j) = \begin{cases} +1 & \text{if } U_i \in pre(l_j) \\ 0 & \text{otherwise.} \end{cases},$$

$$\mathbf{C^{Post}}(U_i, l_j) = \begin{cases} +1 & \text{if } U_i \in post(l_j) \\ 0 & \text{otherwise.} \end{cases}$$

Clearly, the pre activity matrix indicates the pre local derivatives for each labelled activity, i.e. the local derivatives which can fire this activity. The post activity matrix indicates the post local derivatives, i.e. the derived local derivatives after firing a activity. For a labelled activity this should always be a singleton set. The activity matrix is equal to the difference between the pre and post activity matrices, i.e. $\mathbf{C} = \mathbf{C^{Pre}} - \mathbf{C^{Post}}$. An algorithm for automatically deriving the activity matrix and pre activity matrix from any PEPA model was previously presented in [3].

The concepts of P/T net and P/T system originate in Petri net theory ("P/T" signifies "place/transition") but they can also be interpreted in terms of conditions and events.

Definition 4. ***(P/T net, Marking, P/T system**, [1])*

1. *A* Place/Transition net *(P/T net) is a structure* $\mathcal{N} = (P, T, \mathbf{Pre}, \mathbf{Post})$ *where: P and T are the sets of* places *and* transitions *respectively;* **Pre** *and* **Post** *are the $|P| \times |T|$ sized, natural valued,* incidence matrices.

2. *A* marking *is a vector* $\mathbf{m} : P \to \mathbb{N}$ *that assigns to each place of a P/T net a nonnegative integer (number of tokens).*

3. *A* P/T system *is a pair* $\mathcal{S} = \langle \mathcal{N}, \mathbf{m_0} \rangle$*: a net* $\mathcal{N}$ *with an initial marking* $\mathbf{m_0}$*.*

2.2 Place/Transition Structure in PEPA Models

In order to take advantage of the theory developed for P/T systems in the context of PEPA models we must first establish the P/T system corresponding to a given PEPA model. This is straightforward given the definitions presented in the previous subsection.

From Definition 4, it is easy to see that the structure $\mathcal{N} = \left(\mathcal{D}, \mathcal{A}_{\text{label}}, \mathbf{C^{Pre}}, \mathbf{C^{Post}}\right)$ derived from a PEPA model is a P/T net, where $\mathcal{D}, \mathcal{A}_{\text{label}}$ are the sets of all local derivatives and labelled activities of the PEPA model respectively, and $\mathbf{C^{Pre}}, \mathbf{C^{Post}}$ are the pre and post activity matrices respectively. Given a starting state $\mathbf{m_0}$, $\mathcal{S} = \langle \mathcal{N}, \mathbf{m_0} \rangle$ is a P/T system. Clearly, each reachable marking $\mathbf{m}$ from $\mathbf{m_0}$ is a state of the aggregated CTMC underlying the given PEPA model. This leads us to:

Theorem 1. *There is a P/T system underlying any PEPA model, that is* $\langle \mathcal{N}, \mathbf{m_0} \rangle$*, where* $\mathbf{m_0}$ *is the starting state;* $\mathcal{N} = \left(\mathcal{D}, \mathcal{A}_{label}, \mathbf{C}^{Pre}, \mathbf{C}^{Post}\right)$ *is P/T net: where $\mathcal{D}$ is the set of local derivatives,* $\mathcal{A}_{label}$ *is the labelled activity set;* $\mathbf{C}^{Pre}$ *and* $\mathbf{C}^{Post}$ *are the pre and post activity matrices respectively.*

A P/T net, a particular class of Petri net, like PEPA provides a mathematical modelling language for the description of discrete, distributed systems. In [6] Ribaudo has defined a stochastic Petri net semantics for stochastic process algebras, including PEPA. As in our work here, her approach associates each local derivative with a place and each activity with a transition. To cope with the difference between action types and transitions, she defined a labeling function that maps transition names into action names. Similarly, our approach is to attach distinct labels to each action name, as indicated by the definition of labelled activity. However, since Ribaudo's approach does not include aggregation as we do, the mapping semantics in [6] does not help with the state-space explosion problem in structural analysis for large scale PEPA models with repeated components.

Previous work on structural analysis of PEPA models in [4] has some similarities with our approach. However, class of PEPA considered in [4] is somewhat restricted in particular no repeated components are allowed. Moreover, the problem of the difference between actions and transitions is not considered.

2.3 Some terminology

Now we introduce some terminology related to P/T systems for PEPA models (see [1] for reference).

A transition l is *enabled* at a state $\mathbf{m}$ iff $\mathbf{m} \geq \mathbf{C^{Pre}}[\cdot, l]$; its firing yields a new state $\mathbf{m} = \mathbf{m_0} + \mathbf{C^{Pre}}[\cdot, l]$. This fact is denoted by $\mathbf{m} \xrightarrow{l} \mathbf{m}'$. An *occurrence sequence* from $\mathbf{m}$ is a sequence of transitions $\sigma = t_1 \cdots t_k \cdots$ such that $\mathbf{m} \xrightarrow{t_1} \mathbf{m_1} \cdots \xrightarrow{t_k} \mathbf{m_k} \cdots$. The *language* of $\mathcal{S} = \langle \mathcal{N}, \mathbf{m_0} \rangle$, denoted by $L(\mathcal{S})$ or $L(\mathcal{N}, \mathbf{m_0})$, is the set of all the occurrence sequences from the starting state $\mathbf{m}_0$. A state $\mathbf{m}$ is said to be reachable from $\mathbf{m}_0$ if there exists a σ in $L(\mathcal{S})$ such that $\mathbf{m}_0 \xrightarrow{\sigma} \mathbf{m}$, that is $\mathbf{m} = \mathbf{m}_0 + \mathbf{C} \cdot \boldsymbol{\sigma}$, where $\boldsymbol{\sigma}$ is the firing count vector corresponding to σ. The set of all the reachable states from $\mathbf{m}$, called *reachability set* from $\mathbf{m}$, is denoted by $\mathrm{RS}(\mathcal{N}, \mathbf{m})$. According to the definition, the reachability set of the P/T system $\mathcal{S} = \langle \mathcal{N}, \mathbf{m}_0 \rangle$ is

$$\mathrm{RS}(\mathcal{N}, \mathbf{m}_0) = \left\{ \mathbf{m} \in \mathbb{N}^{|P|} \mid \exists \sigma \in L(\mathcal{S}) \text{ such that } \mathbf{m} = \mathbf{m}_0 + \mathbf{C} \cdot \boldsymbol{\sigma} \right\}.$$

where $\boldsymbol{\sigma}$ is the firing count vector of the occurrence sequence σ. Clearly, the reachability set $\mathrm{RS}(\mathcal{N}, \mathbf{m}_0)$ is the state space of the CTMC underlying the given PEPA model starting from $\mathbf{m}_0$. The correspondence between P/T systems and PEPA models is shown in Table 1.

Table 1: P/T structure in PEPA models

P/T terminology	PEPA terminology
P: place set	$\mathcal{D}$: local derivative set
T: transition set	$\mathcal{A}_{\text{label}}$: labelled activity set
Pre: pre- matrix	$\mathbf{C^{Pre}}$: pre- activity matrix
Post: post- matrix	$\mathbf{C^{Post}}$: post- activity matrix
$\mathbf{C} = \mathbf{Pre} - \mathbf{Post}$: incidence matrix	$\mathbf{C} = \mathbf{C^{Pre}} - \mathbf{C^{Post}}$: activity matrix
$\mathbf{m}$: marking	$\mathbf{m}$: state vector
$\mathrm{RS}(\mathcal{N}, \mathbf{m_0})$: reachbility set (from $\mathbf{m_0}$)	$\mathrm{RS}(\mathcal{N}, \mathbf{m_0})$: state space (with starting state $\mathbf{m_0}$)

3 Improved Deadlock-Checking Method for PEPA

A deadlock is a state that cannot fire any activity. This section will present an efficient deadlock checking method which does not heavily suffer the size of the state space.

3.1 Linearisation of state space

According to the definition, the reachability set $\mathrm{RS}(\mathcal{S})$ of a given PEPA model with the activity matrix $\mathbf{C}$ and starting state $\mathbf{m}_0$ is

$$\mathrm{RS}(\mathcal{N}, \mathbf{m_0}) = \left\{ \mathbf{m} \in \mathbb{N}^{|P|} \mid \exists \sigma \in L(\mathcal{S}) \text{ such that } \mathbf{m} = \mathbf{m}_0 + \mathbf{C} \cdot \boldsymbol{\sigma} \right\}.$$

The definition of the reachability set is descriptive rather than constructive, i.e. it does not help us to derive and store the state space. Moreover, we should point out that, for some given $\boldsymbol{\sigma} \in \mathbb{N}^{|P|}$, $\mathbf{m} = \mathbf{m_0} + \mathbf{C}\boldsymbol{\sigma} \in \mathbb{N}^{|P|}$ may not be valid states because there may be no occurrence sequences corresponding to these $\boldsymbol{\sigma}$. However $\mathbf{m}$ is said to belong to a generalisation of the reachability set: the *linearised reachability set*. Before giving these definitions, we first give the analogies for *flow* and *semiflow* in the context of PEPA. (For the definitions of these concept in the context of P/T system, see [1].)

Definition 5. *(**Flow, Semiflow, and Consistent**) Let C be the activity matrix of a given PEPA model.*

1. *A* p-flow *is a vector $\mathbf{y} : P \to \mathbb{Q}$ such that $\mathbf{y}^T\mathbf{C} = 0$. Natural and nonnegative flows are called* semiflows*: vectors $\mathbf{y} : P \to \mathbb{N}$ such that $\mathbf{y}^T\mathbf{C} = 0$.*

2. *A* basis *(respectively,* fundamental set*) of p-flows (respectively p-semiflows), $\mathbf{B} = \{\mathbf{y_1}, \mathbf{y_2}, \cdots, \mathbf{y_q}\}$ (respectively, $\mathbf{\Phi} = \{\mathbf{y_1}, \mathbf{y_2}, \cdots, \mathbf{y_q}\}$) is a minimal set which will generate any p-flow (respectively, p-semiflow) as follows: $\mathbf{y} = \sum_{\mathbf{y}_j \in \mathbf{\Psi}} k_j \mathbf{y}_j$, $k_j \in \mathbb{Q}$.*

3. *A* t-flow *is a vector $\mathbf{x} : P \to \mathbb{Q}$ such that $\mathbf{Cx} = 0$. Natural and nonnegative flows are called* semiflows*: vectors $\mathbf{x} : P \to \mathbb{N}$ such that $\mathbf{Cx} = 0$. A model is* consistent *if there exists a t-semiflow whose support covers P.*

Obviously, p-semiflow is a special kind of p-flow while t-semiflow is a special t-flow. Let $\mathbf{B}$ and $\mathbf{\Phi}$ be bases of p-flows and fundamental sets of p-semiflows respectively. Then for any $\mathbf{m} \in \mathrm{RS}(\mathcal{N}, \mathbf{m_0})$, we have $\mathbf{Bm} = 0$ and $\mathbf{\Phi m} = 0$. This does not imply that any $\mathbf{m} \in \mathbb{N}^{|P|}$ that satisfies $\mathbf{Bm} = 0$ or $\mathbf{\Phi m} = 0$ is in $\mathrm{RS}(\mathcal{N}, \mathbf{m_0})$. However, they do belong to generalised reachability sets:

Definition 6. *(**Linearised Reachability Set**, [7]) Let $\mathcal{S}$ be a P/T system.*

1. *Its* linearised reachability set *using the* state equation *is defined as*

$$\mathrm{LRS}^{\mathrm{SE}}(\mathcal{S}) = \left\{\mathbf{m} \in \mathbb{N}^{|P|} \mid \exists \boldsymbol{\sigma} \in \mathbb{N}^{|T|} \text{ such that } \mathbf{m} = \mathbf{m_0} + \mathbf{C} \cdot \boldsymbol{\sigma}\right\}.$$

2. *Its* linearised reachability set *using the* state equation over reals *is defined as*

$$\mathrm{LRS}^{\mathrm{SER}}(\mathcal{S}) = \left\{\mathbf{m} \in \mathbb{N}^{|P|} \mid \exists \boldsymbol{\sigma} \geq \mathbf{0} \text{ such that } \mathbf{m} = \mathbf{m_0} + \mathbf{C} \cdot \boldsymbol{\sigma}\right\}.$$

3. *Its* linearised reachability set *using a* basis B of P-flows *is defined as*

$$\mathrm{LRS}^{\mathrm{Pf}}(\mathcal{S}) = \left\{\mathbf{m} \in \mathbb{N}^{|P|} \mid \mathbf{B} \cdot \mathbf{m} = \mathbf{B} \cdot \mathbf{m_0}\right\}.$$

4. *Its* linearised reachability set *using a* the fundamental set of P-semiflows *is defined as*

$$\mathrm{LRS}^{\mathrm{Psf}}(\mathcal{S}) = \left\{\mathbf{m} \in \mathbb{N}^{|P|} \mid \mathbf{\Phi} \cdot \mathbf{m} = \mathbf{\Phi} \cdot \mathbf{m_0}\right\}.$$

It is obvious that $RS(\mathcal{S}) \subseteq LRS^{SE}(\mathcal{S}) \subseteq LRS^{SER}(\mathcal{S}) \subseteq LRS^{Pf}(\mathcal{S}) \subseteq LRS^{Psf}(\mathcal{S})$.

3.2 New Deadlock-checking method

This subsection presents an equivalent deadlock-checking theorem and deadlock-checking algorithm. The theorem is presented without proof but details can be found in [2]. We first introduce the concept of equal conflict (see [1]).

Definition 7. *(**Equal Conflict**) Let $\mathcal{N} = (P, T, Pre, Post)$ be a P/T net. $\mathcal{N}$ is called* equal conflict *(EQ), if $pre(l) \cap pre(l') \neq \emptyset$ implies $\mathbf{Pre}[\cdot, l] = \mathbf{Pre}[\cdot, l']$.*

In the context of PEPA this can be interpreted as *the P/T structure underlying a PEPA model is EQ if and only if for any two labelled activities l and l', their pre sets are either equal or distinct, i.e. either $pre(l) = pre(l')$ or $pre(l) \cap pre(l') = \emptyset$.*

Now we state our equivalent deadlock-checking theorem.

Theorem 2. *If the P/T system $\mathcal{S}$ underlying a PEPA model is a consistent, EQ system, then*

1. $\mathrm{LRS}^{\mathrm{SE}}(\mathcal{S})$ *is deadlock-free* $\Longleftrightarrow$ $\mathrm{RS}(\mathcal{S})$ *is deadlock-free.*
2. $\mathrm{LRS}^{\mathrm{SER}}(\mathcal{S})$ *is deadlock-free* $\Longleftrightarrow$ $\mathrm{RS}(\mathcal{S})$ *is deadlock-free.*
3. $\mathrm{LRS}^{\mathrm{Pf}}(\mathcal{S})$ *is deadlock-free* $\Longleftrightarrow$ $\mathrm{RS}(\mathcal{S})$ *is deadlock-free.*
4. $\mathrm{LRS}^{\mathrm{Psf}}(\mathcal{S})$ *is deadlock-free* $\Longleftrightarrow$ $\mathrm{RS}(\mathcal{S})$ *is deadlock-free.*

According to Theorem 2, for a consistent, EQ system $\mathcal{S}$, to tell whether $\mathrm{RS}(\mathcal{S})$ has deadlocks it is sufficient to check whether $\mathrm{LRS}^{\mathrm{Psf}}(\mathcal{S})$ has deadlocks. As we mentioned, the activity l is disabled at $\mathbf{m}$ means that there exists a U such that $\mathbf{m}[U] < \mathbf{C}^{\mathbf{Pre}}[U, l]$. Note that

$$\mathbf{m}[U] < \mathbf{C}^{\mathbf{Pre}}[U, l] \Longleftrightarrow \mathbf{m}[U] = 0 \text{ and } \mathbf{C}^{\mathbf{Pre}}[U, l] = 1.$$

Thus only states $\mathbf{m}$ with zeros in some particular places can possibly have a deadlock. Based on this idea, we provide a deadlock-checking algorithm, see Algorithm 1.

Algorithm 1 Deadlock-checking in $\mathrm{LRS}^{\mathrm{Psf}}$

1: **for all** $l \in \mathcal{A}_{\mathrm{label}}$ **do**
2: **if** l is an individual activity **then**
3: $K(l) = \{\mathbf{m} \in \mathbb{N}^{|\mathcal{D}|} \mid \mathbf{m}[U] = 0, \mathbf{C}^{\mathbf{Pre}}[U, l] = 1\}$ // where $\{U\} = \mathrm{pre}(l)$
4: **else if** l is a shared activity **then**
5: $K(l) = \bigcup_{U \in \mathrm{pre}(l)} \{\mathbf{m} \in \mathbb{N}^{|\mathcal{D}|} \mid \mathbf{m}[U] = 0, \mathbf{C}^{\mathbf{Pre}}[U, l] = 1\}$
6: **end if**
7: **end for**
8: $K = \bigcap_{l \in \mathcal{A}_{\mathrm{label}}} K(l)$
9: If $K \bigcap \mathrm{LRS}^{\mathrm{Psf}} = \emptyset$, then $\mathrm{LRS}^{\mathrm{Psf}}$ is deadlock-free. Otherwise, $\mathrm{LRS}^{\mathrm{Psf}}$ at least has one deadlock.

Our deadlock-checking algorithm is structure- or equation-based, rather than state-space-based, so it avoids searching the entire state space and thus avoids the state-space explosion problem. Although Theorem 2 requires the conditions of consistent and EQ, Algorithm 1 is free from these restrictions since it deals with the linearised state space. That means, for any general PEPA model with or without the consistent and EQ restrictions, if the generalised state space has no deadlocks reported by using Algorithm 1, then the model has no deadlocks. But if it reports deadlocks in the generalised state space, it cannot tell whether there is a deadlock in the model, except for a consistent and EQ model. The weakness of this approach is that some symbolic computation may be needed. But at this cost, a non-negligible advantage has been obtained: this method can tell when or how a system structure may have deadlocks.

Table 2: Activity matrix of Model 1

	$task_1$	$task_2$	
$User_1$	-1	1	
$User_2$	1	-1	
$Provider_1$	-1	1	
$Provider_2$	1	-1	

3.3 Example

Now we consider an example PEPA model which has a consistent and EQ P/T structure.

$$\begin{aligned}
User_1 &\stackrel{def}{=} (task_1, 1).User_2 \\
User_2 &\stackrel{def}{=} (task_2, 1).User_1 \\
Provider_1 &\stackrel{def}{=} (task_1, 1).Provider_2 \\
Provider_2 &\stackrel{def}{=} (task_2, 1).Provider_1 \\
Model\ 1 &\stackrel{def}{=} (User_1[M_1] \parallel User_2[M_2]) \underset{\{task_1, task_2\}}{\bowtie} (Provider_1[N_1] \parallel Provider_2[N_2]).
\end{aligned}$$

Table 2 lists the activity matrix of Model 1. The activity matrix $\mathbf{C}$ and pre activity matrix $\mathbf{C^{Pre}}$ are listed below:

$$\mathbf{C} = \begin{pmatrix} -1 & 1 \\ 1 & -1 \\ -1 & 1 \\ 1 & -1 \end{pmatrix}, \quad \mathbf{C^{Pre}} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 0 \\ 0 & 1 \end{pmatrix}.$$

First, let us determine $\mathrm{LRS}^{\mathrm{Psf}}(\mathcal{S})$. Solving $\mathbf{C}^T\mathbf{y} = 0$, we get a basis of the solution space which forms the rows of Φ:

$$\mathbf{\Phi} = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix}.$$

Notice $\mathbf{m_0} = (M_1, M_2, N_1, N_2)^T$, so

$$\begin{aligned}
\mathrm{LRS}^{\mathrm{Psf}}(\mathcal{S}) =& \{\mathbf{m} \in \mathbb{N}^4 \mid \Phi\mathbf{m} = \mathbf{\Phi m_0}\} \\
=& \left\{ \mathbf{m} \in \mathbb{N}^4 \middle| \begin{array}{r} \mathbf{m}[User_1] + \mathbf{m}[User_2] = M_1 + M_2; \\ \mathbf{m}[User_2] + \mathbf{m}[Provider_1] = M_2 + N_1; \\ \mathbf{m}[Provider_1] + \mathbf{m}[Provider_2] = N_1 + N_2 \end{array} \right\}
\end{aligned}$$

Note that each of the semiflows corresponds to an invariant of the model. The first and third express the fact that the number of users, and the number of providers respectively, is constant within the model. The second expresses the coupling between the components, i.e. the cooperations ensure that the numbers of local derivatives in the two components always change together.

Secondly, we determine the potential deadlock set K for each activity. According to Algorithm 1,

$$K(task1) = \{\mathbf{m} \mid \mathbf{m}[User_1] = 0 \text{ or } \mathbf{m}[Provider_1] = 0\},$$

$$K(task2) = \{\mathbf{m} \mid \mathbf{m}[User_2] = 0 \text{ or } \mathbf{m}[Provider_2] = 0\},$$

$$
\begin{aligned}
K =& K(task1) \cap K(task2) \\
=& \{\mathbf{m} \mid \mathbf{m}[User_1] = 0, \mathbf{m}[User_2] = 0\} \cup \{\mathbf{m} \mid \mathbf{m}[User_1] = 0, \mathbf{m}[Provider_2] = 0\} \\
& \cup \{\mathbf{m} \mid \mathbf{m}[Provider_1] = 0, \mathbf{m}[User_2] = 0\} \cup \{\mathbf{m} \mid \mathbf{m}[Provider_1] = 0, \mathbf{m}[Provider_2] = 0\}
\end{aligned}
$$

Finally, the deadlock set in $\mathrm{LRS}^{\mathrm{Psf}}$ is

$$
\begin{aligned}
& K \cap \mathrm{LRS}^{\mathrm{Psf}} \\
& = \left\{ \mathbf{m} \in \mathbb{N}^4 \;\middle|\; \begin{array}{l} \mathbf{m}[User_1] + \mathbf{m}[User_2] = M_1 + M_2; \\ \mathbf{m}[User_2] + \mathbf{m}[Provider_1] = M_2 + N_1; \\ \mathbf{m}[Provider_1] + \mathbf{m}[Provider_2] = N_1 + N_2 \end{array} \right\} \\
& \quad \bigcap \{\mathbf{m} \mid (\mathbf{m}[User_1] = \mathbf{m}[Provider_2] = 0) \vee (\mathbf{m}[Provider_1] = \mathbf{m}[User_2] = 0)\} \\
& = \left\{ \mathbf{m} \in \mathbb{N}^4 \mid \begin{array}{l} (\mathbf{m} = (0, M_1 + M_2, N_1 + N_2, 0)^T \wedge M_1 + N_2 = 0) \\ \vee (\mathbf{m} = (M_1 + M_2, 0, 0, N_1 + N_2)^T \wedge M_2 + N_1 = 0) \end{array} \right\} \\
& = \left\{ \mathbf{m} \in \mathbb{N}^4 \mid \begin{array}{l} (\mathbf{m} = (0, M_1 + M_2, N_1 + N_2, 0)^T \wedge M_1 = N_2 = 0) \\ \vee (\mathbf{m} = (M_1 + M_2, 0, 0, N_1 + N_2)^T \wedge M_2 = N_1 = 0) \end{array} \right\}.
\end{aligned}
$$

In other words, for Model 1 with $\mathbf{m_0} = (M_1, M_2, N_1, N_2)^T$, only when $M_1 = N_2 = 0$ or $M_2 = N_1 = 0$, $K \cap \mathrm{LRS}^{\mathrm{Psf}} \neq \emptyset$, i.e. the system has at least one deadlock. Otherwise, the system is deadlock-free as long as $M_1 + N_2 \neq 0$ and $M_2 + N_1 \neq 0$.

This example illustrates that our deadlock-checking method can not only tell whether a particular system is deadlock-free but also how a system structure may lead to deadlocks.

4 Summary

This paper has revealed the P/T structure underlying PEPA models. Based on techniques developed for P/T systems, we proposed a new deadlock-checking algorithm for PEPA models, which can efficiently reduce the computational complexity of deadlock-checking and avoid the state-space explosion problem.

References

[1] J. M. Colom, E. Teruel, and M. Silva. Logical properties of P/T system and their analysis. MATCH Summer School (Spain), Septemper 1998.

[2] Jie Ding. *Structural and Fluid Analysis of Large Scale PEPA models — with Applications to Content Adaptation Systems*. PhD thesis, The Univeristy of Edinburgh, 2009.

[3] Jie Ding and Jane Hillston. Consistency and convergence of fluid approximations of PEPA models. Technical report, The University of Edinburgh, UK, July 2008.

[4] Stephen Gilmore, Jane Hillston, and Laura Recalde. Elementary structural analysis for PEPA. Technical report, The University of Edinburgh, UK, December 1997.

[5] J. Hillston. Fluid flow approximation of PEPA models. In *International Conference on the Quantitative Evaluation of Systems (QEST'05)*. IEEE CS Press, 2005.

[6] Marina Ribaudo. Stochastic Petri net semantics for stochastic process algebras. In *Proceedings of the Sixth International Workshop on Petri Nets and Performance Models*, Washington DC, USA, 1995. IEEE Computer Society.

[7] M. Silva, E. Teruel, and J. M. Colom. Linear algebraic and linear programming techniques for the analyisis of place/transition net systems. In *Lecture Notes in Computer Science*, volume 1491. Springer-Verlag, 1996.

A general result for deriving product-form solutions in Markovian models

Andrea Marin
Università Ca' Foscari di Venezia

Maria Grazia Vigliotti
Imperial College London

Abstract

In this paper we provide a general method to derive product-form solutions for stochastic models. We take inspiration from the Reversed Compound Agent Theorem [3] and we provide new result using labeled Markov automata. Our result is a generalization of RCAT which encompasses a bigger class of product-form solutions and its proof is based on solving the global balance equations.

1 Introduction

Product-form solutions have been extensively studied for different systems in the theory of Markovian stochastic models. The holy grail in this field consists in finding general conditions that characterize (most of) the systems that have product-form solutions. In this direction a lot of work has been done using Generalized Stochastic Petri Nets (GSPNs) [1] and Performance Evaluation Process Algebra (PEPA) [6, 2, 8, 3, 4]. In this paper we use a variant of *labeled stochastic automata* to study product-form solutions. Our starting point is the Reversed Compound Agent Theorem (RCAT)[3] and we expand that work in different directions:

- There are some limitations in the original formulation of RCAT [3]. To show this we have produced an example in [7] which shows that a model exists that enjoys the product-form solution but the structural conditions of RCAT, for such model, do not hold. The logical conclusion is that more general conditions exist that characterize a larger class of systems that enjoy product-form solutions.
- We provide a new, entirely different proof of the theorem (with respect to that formulated for RCAT) based on the global balance equation analysis. The proofs in the original paper [3] and all their subsequents were based on the application of Kolmogorov's criteria.

We have decided to depart from the original formulation of the theorem in terms of PEPA in favour of the new formulation based on labeled automata since:

1. The definition of labelled Markov automata retains the compositionally of process algebra, and allows for modular descriptions of systems.
2. Labelled Markov automata is a general enough framework for comparing different product form results specified in various formalisms such as queueing networks, stochastic Petri nets, PEPA etc.
3. Our new proof, which is formulated using global balance equations, is naturally cast in terms of labelled Markov automata.

2 Labeled Markov automata (LMA)

We assume the reader familiar with probability theory and the basics of CTMCs. In this section we provide the basic definitions that will allow us to define the cooperation among CTMCs.

Definition 2.1 (Labeled automaton) *A Markov automaton is a tuple $\mathcal{M} = \langle \mathsf{S}, Act, \rightarrow \rangle$ such that:*

1. S *is the denumerable set of states with $s_1, s_2, \ldots s_n, \ldots$ range over it,*
2. *Act is the set of* action labels *with $a, b, \ldots$ range over it,*
3. $\rightarrow$ *is the transition relation between states defined as $\rightarrow$: $\mathsf{S} \times Act \times (\mathbb{R}^+ \cup \mathbf{Var}) \times \mathsf{S}$, where $\mathbb{R}^+$ is the set of positive real numbers and $\mathbf{Var}$ is the set of variable names.*

For readability, we write $(s_1, a, \lambda, s_1') \in \rightarrow$ as $s_1 \xrightarrow{a,\lambda} s_1'$.

Initially, one can think LMA as a CTMC where the transitions have been labeled. The reader might be puzzled by the use of labels, their rôle will become apparent in the following definition of interactive Markov automata, where the labels will help in defining which actions should co-operate and which should not.

Definition 2.2 (Interacting LMAs) *Let $\mathcal{M}_1 = \langle \mathsf{S}_1, Act_1, \rightarrow_1 \rangle$ and $\mathcal{M}_2 = \langle \mathsf{S}_2, Act_2, \rightarrow_2 \rangle$ be two LMAs. The* interacting *LMA $\mathcal{M}_1 \oplus_L \mathcal{M}_2 = \langle \mathsf{S}, Act, \rightarrow, \rangle$ with $L \subseteq Act_1 \cap Act_2$ is a new automata defined as follows:*

1. $\mathsf{S} = \mathsf{S}_1 \times \mathsf{S}_2$.
2. $Act = Act_1 \cup Act_2$.
3. $\rightarrow$ *is the smallest relation defined by the rules below:*

$$\frac{s_1 \xrightarrow{a,\lambda}_1 s_1' \qquad s_2 \xrightarrow{a,x_a}_2 s_2'}{(s_1, s_2) \xrightarrow{a,\lambda} (s_1', s_2')} (a \in L)$$

$$\frac{s_1 \xrightarrow{a,r}_1 s_1'}{(s_1, s_2) \xrightarrow{a,r} (s_1', s_2)} (a \notin L)$$

We have omitted the symmetric rules. We reserve the Greek letters to range over $\mathbb{R}^+$ and the Roman u, q, p over $\mathbb{R}^+ \cup \mathbf{Var}$ and the letter $x_a, y_a, z_a \ldots$ to range over $\mathbf{Var}$. We call the set of labels L the cooperation set. The set of active actions $\mathcal{A}(\mathcal{M})$ is defined as: $a \in \mathcal{A}(\mathcal{M})$ if, for some s, $s \xrightarrow{a,\lambda} s'$ with $\lambda \in \mathbb{R}^+$. The set of passive actions $\mathcal{P}(\mathcal{M})$ is defined as: $a \in \mathcal{P}(\mathcal{M})$ if, for some s, $s \xrightarrow{a,x_a} s'$, with $x_a \in \mathbf{Var}$.

The definitions above show that LMAs can perform more transitions than labeled CTMCs. First of all, transitions are divided into active and passive. Cooperation between automata happens only between an active and a passive action, never between two active actions or two passive actions. In the cooperation the unspecified rate moves at the speed of the automaton with the active rate. The meaning of passive transition is directly inspired by PEPA [5], instead of using the symbol $\top$ we use variables for convenience. Note that our cooperation is more restrictive with respect to the one defined in PEPA, yet fully adequate to the purpose

of our work. Generalization of cooperating automata that deals with active-active transitions is possible, but outside the scope of our work. If an automaton does not contain any passive transition, then the underlying model description is a CTMC. To see this it suffices to associate to each transition $s_1 \xrightarrow{a,\lambda} s_2$ a random variable $X_{a,\lambda}$ such that $\mathbb{P}(X_{a,\lambda} \leq t) = 1 - e^{-\lambda t}$. On this basis we justify the following definitions.

Definition 2.3 (Open and closed automata) *We distinguish the following classes of automata:*

1. *A LMA $\mathcal{M} = \langle \mathsf{S}, Act, \rightarrow \rangle$ is called* open *if there exists a label $a \in Act$ and a state $s \in \mathsf{S}$ such that a passively enabled in s, i.e., $\exists s' \in \mathsf{S}$ such that $s \xrightarrow{a,x_a} s'$ and $s \neq s'$.*
2. *A LMA $\mathcal{M} = \langle \mathsf{S}, Act, \rightarrow \rangle$ is called* closed *if it is not open.*

Given a closed LMA $\mathcal{M}$ all the transitions are carried out according to an exponentially distributed random delay. If more than one transition is possible from a state s a probabilistic choice will occurs. It is easy to show that the sojourn time in each state of any LMA is exponentially distributed and that the underlying time-homogenuous CTMC is derived in the usual manner [5]. The notation for the derivation of the CTMC here is consistent with [5]. Once established how to derive the CTMC we can talk directly about the properties of the LMA, meaning those properties of the CTMC. The analysis of LMAs modeling power is out of the scope of this paper, thus we introduce a set of restrictions that simplifies the presentation of the theoretical result about the product-form solutions of cooperating LMAs. It should be pointed out that although these restrictions limit the flexibility of LMA modeling, they do not reduce the applicability of the results that will follow. In practice, we require a *well-formed* LMA $\mathcal{M} = \langle \mathsf{S}, Act, \rightarrow \rangle$ to satisfy the following properties:

1. given a label $a \in Act$ then all the transitions labeled by a are either active or passive: $\mathcal{A}(\mathcal{M}) \cap \mathcal{P}(\mathcal{M}) = \emptyset$
2. if a is a passive label, then for every state s of the automaton there exists only one transition labeled by a outgoing from s i.e. $\forall s \exists s' \in \mathsf{S}, \quad s \xrightarrow{a,x_a} s' \wedge \forall s, s', s'' \in \mathsf{S},\ s \xrightarrow{a,x_a} s' \wedge s \xrightarrow{a,x_a} s'' \implies s' = s''$

For the rest of the paper we assume to work with well-formed Markov automata.

3 Product-form solutions

Let us consider two (well-formed) automata $\mathcal{M}_1$ and $\mathcal{M}_2$. We are interested in expressing the steady state distribution of $\mathcal{M}_1 \oplus_L \mathcal{M}_2$ as the steady state distribution of each component. Before introducing the main theorem we define a last operation on the automata, i.e., the closure with respect to a label.

Definition 3.1 (Closure of an automaton) *Let $\mathcal{M} = \langle \mathsf{S}, Act, \rightarrow_{\mathcal{M}} \rangle$ be a LMA and $a \in \mathcal{P}(\mathcal{M})$, then the* closure *of $\mathcal{M}$, written $\mathcal{M}\{a \leftarrow \lambda\}$, is defined as $\mathcal{M}\{a \leftarrow \lambda\} = \langle \mathsf{S}, Act, \rightarrow_{\mathcal{M}\{a \leftarrow \lambda\}} \rangle$, where*

$$\rightarrow_{\mathcal{M}\{a \leftarrow \lambda\}} = \{(s_i, b, t, s_j) : (s_i, b, t, s_j) \in \rightarrow_{\mathcal{M}} \wedge b \neq a \wedge t \in \mathbb{R}^+\} \cup \{(s_i, a, \lambda, s_j) : (s_i, a, x_a, s_j) \in \rightarrow_{\mathcal{M}}\}$$

Several closures may be specified in a compact way, e.g., let $a, b \in \mathcal{P}(\mathcal{M})$, then $\mathcal{M}\{x_a \leftarrow \lambda_1, x_b \leftarrow \lambda_2\}$ corresponds to $(\mathcal{M}\{x_a \leftarrow \lambda_1\})\{x_b \leftarrow \lambda_2\}$

In the following theorem we use $a_1, a_2, \ldots$ to denote the labels, and $x_1, x_2, \ldots$ to denote the variables (instead of $x_{a_1}, x_{a_2} \ldots$).

Theorem 3.2 *Let $\mathcal{M}_1$ and $\mathcal{M}_2$ be two well-formed LMAs that cooperate on a finite set of labels $L = \{a_1, \ldots, a_n\}$, such that $\mathcal{M}_1 \oplus_L \mathcal{M}_2$ is ergodic.*

If there exists the set of rates $\{\lambda_1, \ldots, \lambda_n\} \neq \emptyset$ which satisfies the following equations:

$$\forall s_k \in \mathsf{S}_1, \forall a_i \in \mathcal{A}(\mathcal{M}_1) \qquad \frac{\sum_{s_j \in \mathsf{S}_1} \mathbf{q}(s_j, a_i, s_k)\pi_1(s_j)}{\pi_1(s_k)} = \lambda_i \tag{1}$$

or

$$\forall s_k \in \mathsf{S}_2, \forall a_i \in \mathcal{A}(\mathcal{M}_2) \qquad \frac{\sum_{s_j \in \mathsf{S}_2} \mathbf{q}(s_j, a_i, s_k)\pi_2(s_j)}{\pi_2(s_k)} = \lambda_i \tag{2}$$

where π_1 and π_2 are the stationary probability distributions of the closed automata $\mathcal{M}_1^\dagger$ and $\mathcal{M}_2^\dagger$:

$$\mathcal{M}_1^\dagger = \mathcal{M}_1\{a_i \leftarrow \lambda_i : a_i \in \mathcal{P}(\mathcal{M}_1)\} \quad \mathcal{M}_2^\dagger \quad = \quad \mathcal{M}_2\{a_i \leftarrow \lambda_i : a_i \in \mathcal{P}(\mathcal{M}_2)\}$$

then the steady-state solution of $\mathcal{M}_1 \oplus_L \mathcal{M}_2$ has the product-form:

$$\pi(\mathcal{M}_1 \oplus_L \mathcal{M}_2) \quad \propto \quad \pi_1(\mathcal{M}_1^\dagger)\pi_2(\mathcal{M}_2^\dagger) \tag{3}$$

where π_i is the steady state distribution of $\mathcal{M}_i$, with $i = 1, 2$.

Note that Equations (1) and (2) basically say that the total flow incoming into a state due to the active transitions labeled by a must be proportional to the stationary probability of that state in the closure of the automaton.

In what follows we present the proof assuming that automata synchronise on label 'a' only. This is only for readability. The proof with any number of synchronizing labels is a simple generalization of the one presented below.

Proof 1 *Without loss of generality we assume that a is active in $\mathcal{M}_1$ and passive in $\mathcal{M}_2$. The global balance equations (GBEs) for $\mathcal{M}_1$ and $\mathcal{M}_2^\dagger$ are:*

$$\pi_{\mathcal{M}_1}(r)\Big(\sum_{r' \in \mathsf{S}_{\mathcal{M}_1}} \mathbf{q}_{\mathcal{M}_1}(r, a, r') + \sum_{\substack{r' \in \mathsf{S}_{\mathcal{M}_1} \\ b \neq a}} \mathbf{q}_{\mathcal{M}_1}(r, b, r')\Big) = \mathbf{q}_{\mathcal{M}_1}(r', a, r)\pi_{\mathcal{M}_1}(r') + \sum_{\substack{r' \in \mathsf{S}_{\mathcal{M}_1} \\ b \neq a}} \mathbf{q}_{\mathcal{M}_1}(r', b, r)\pi_{\mathcal{M}_1}(r') \tag{4}$$

$$\pi_{\mathcal{M}_2^\dagger}(s)\Big(\underbrace{\mathbf{q}_{\mathcal{M}_2^\dagger}(s, a, s')}_{\lambda} + \sum_{\substack{s' \in \mathsf{S}_{\mathcal{M}_2^\dagger} \\ b \neq a}} \mathbf{q}_{\mathcal{M}_2^\dagger}(s, b, s')\Big) = \sum_{\substack{s' \in \mathsf{S}_{\mathcal{M}_2^\dagger} \\ b \neq a}} \mathbf{q}_{\mathcal{M}_2^\dagger}(s', b, s)\pi_{\mathcal{M}_2^\dagger}(s') + \sum_{s' \in \mathsf{S}_{\mathcal{M}_2^\dagger}} \underbrace{\mathbf{q}_{\mathcal{M}_2^\dagger}(s', a, s)}_{\lambda} \pi_{\mathcal{M}_2^\dagger}(s') \tag{5}$$

where λ is the rate that we substitute into the passive transition of $\mathcal{M}_2$ to make $\mathcal{M}_2^\dagger$. The GBE for the joint state space $(r,s) \in \mathsf{S}_{\mathcal{M}_1} \times \mathsf{S}_{\mathcal{M}_2}$ is:

$$\pi\big((r,s)\big)\Big(\sum_{\substack{r'\in\mathsf{S}_{\mathcal{M}_1}\\ b\neq a}} \mathbf{q}((r,s),b,(r',s)) + \sum_{\substack{s'\in\mathsf{S}_{\mathcal{M}_2}\\ b\neq a}} \mathbf{q}((r,s),b,(r,s')) + \sum_{(r',s')\in\mathsf{S}_{\mathcal{M}_1}\times\mathsf{S}_{\mathcal{M}_2}} \mathbf{q}((r,s),a,(r',s'))\Big) =$$
$$\sum_{\substack{r'\in\mathsf{S}_{\mathcal{M}_1}\\ b\neq a}} \mathbf{q}((r',s),b,(r,s))\pi\big((r',s)\big) + \sum_{\substack{s'\in\mathsf{S}_{\mathcal{M}_2}\\ b\neq a}} \mathbf{q}((r,s'),b,(r,s))\pi\big((r,s')\big)$$
$$+ \sum_{(r',s')\in\mathsf{S}_{\mathcal{M}_1}\times\mathsf{S}_{\mathcal{M}_2}} \mathbf{q}((r',s'),a,(r,s_i))\pi\big((r',s')\big) \quad (6)$$

We substitute product form and we rewrite the rates in each term with the rates of each automaton taking into account self loops:

$$\pi_{\mathcal{M}_1}(r)\pi_{\mathcal{M}_2^\dagger}(s)\Big(\sum_{\substack{r'\in\mathsf{S}_{\mathcal{M}_1}\\ b\neq a}} \mathbf{q}_{\mathcal{M}_1}(r,b,r') + \sum_{\substack{s'\in\mathsf{S}_{\mathcal{M}_2^\dagger}\\ b\neq a}} \mathbf{q}_{\mathcal{M}_2^\dagger}(s,b,s') + \sum_{(r,s)^{a,\mathbf{q}(r,a,r')}(r',s')} \mathbf{q}_{\mathcal{M}_1}(r,a,r') +$$
$$\sum_{(r,s)^{a,\mathbf{q}(r,a,r)}(r,s')} \mathbf{q}_{\mathcal{M}_1}(r,a,r)\Big) = \sum_{\substack{r'\in\mathsf{S}_{\mathcal{M}_1}\\ b\neq a}} \mathbf{q}_{\mathcal{M}_1}(r',b,r)\pi_{\mathcal{M}_1}(r')\pi_{\mathcal{M}_2^\dagger}(s) + \sum_{\substack{s'\in\mathsf{S}_{\mathcal{M}_2^\dagger}\\ b\neq a}} \mathbf{q}_{\mathcal{M}_2^\dagger}(s',b,s)\pi_{\mathcal{M}_1}(r)\pi_{\mathcal{M}_2^\dagger}(s') +$$
$$\sum_{(r',s')^{a,\mathbf{q}(r',a,r)}(r,s)} \mathbf{q}_{\mathcal{M}_1}(r',a,r)\pi_{\mathcal{M}_1}(r')\pi_{\mathcal{M}_2^\dagger}(s') + \sum_{(r,s')^{a,\mathbf{q}(r,a,r)}(r,s)} \mathbf{q}_{\mathcal{M}_1}(r,a,r)\pi_{\mathcal{M}_1}(r)\pi_{\mathcal{M}_2^\dagger}(s')$$

After a few algebraic manipulations substituting the right part of equation (4) we obtain:

$$\mathbf{q}_{\mathcal{M}_1}(r',a,r)\frac{\pi_{\mathcal{M}_1}(r')}{\pi_{\mathcal{M}_1}(r)} + \sum_{(r,s)^{a,\mathbf{q}(r,a,r)}(r,s')} \mathbf{q}_{\mathcal{M}_1}(r,a,r) + \sum_{\substack{s'\in\mathsf{S}_{\mathcal{M}_2^\dagger}\\ b\neq a}} \mathbf{q}_{\mathcal{M}_2^\dagger}(s,b,s') =$$
$$\sum_{\substack{s'\in\mathsf{S}_{\mathcal{M}_2^\dagger}\\ b\neq a}} \mathbf{q}_{\mathcal{M}_2^\dagger}(s',b,s)\frac{\pi_{\mathcal{M}_2^\dagger}(s')}{\pi_{\mathcal{M}_2^\dagger}(s)} + \sum_{(r',s')^{a,\mathbf{q}(r',a,r)}(r,s)} \mathbf{q}_{\mathcal{M}_1}(r',a,r)\frac{\pi_{\mathcal{M}_1}(r')\pi_{\mathcal{M}_2^\dagger}(s')}{\pi_{\mathcal{M}_2^\dagger}(s)\pi_{\mathcal{M}_1}(r)} +$$
$$\sum_{(r,s')^{a,\mathbf{q}(r,a,r)}(r,s)} \mathbf{q}_{\mathcal{M}_1}(r,a,r)\frac{\pi_{\mathcal{M}_2^\dagger}(s')}{\pi_{\mathcal{M}_2^\dagger}(s)}$$

By observing that λ is the reversed rate and with further algebraic manipulations we obtain the following.

$$\pi_{\mathcal{M}_2^\dagger}(s)(\mathbf{q}_{\mathcal{M}_2^\dagger}(s,a,s') + \sum_{(r,s)^{a,\mathbf{q}(r,a,r)}(r,s')} \mathbf{q}_{\mathcal{M}_1}(r,a,r) + \sum_{\substack{s'\in\mathsf{S}_{\mathcal{M}_2^\dagger}\\ b\neq a}} \mathbf{q}_{\mathcal{M}_2^\dagger}(s,b,s')) =$$
$$\sum_{\substack{s'\in\mathsf{S}_{\mathcal{M}_2^\dagger}\\ b\neq a}} \mathbf{q}_{\mathcal{M}_2^\dagger}(s',b,s)\pi_{\mathcal{M}_2^\dagger}(s') + \sum_{s'\in\mathsf{S}_{\mathcal{M}_2^\dagger}} \mathbf{q}_{\mathcal{M}_2^\dagger}(s',a,s)\pi_{\mathcal{M}_2^\dagger}(s') + \sum_{(r,s')^{a,\mathbf{q}(r,a,r)}(r,s)} \mathbf{q}_{\mathcal{M}_1}(r,a,r)\pi_{\mathcal{M}_2^\dagger}(s')$$

This latter can be derived from (5).

In the proof above self-loops were considered. While self-loops do not change the behavior of the Markov Chain, i.e., they have no effect on the global balance equations of a single automaton, they are important in the definition of the structure of the interacting LMA.

In this paragraph we aim to point out the main difference between Theorem 3.2 and RCAT. Our notion of well-formed automaton reflects a structural condition of RCAT, i.e., for each passive label a, exactly one passive transition outgoes from every state. Moreover, RCAT requires that:

$$\forall s_k \in \mathsf{S}_1, \forall a_i \in \mathcal{A}(\mathcal{M}_1) \quad \frac{\mathbf{q}(s_j, a_i, s_k)\pi_1(s_j)}{\pi_1(s_k)} = \lambda_i,$$

where s_j is the *only* state in $\mathcal{M}_1^\dagger$ with an active transition labeled by a_i going into that state s_k. If we observe that λ_i is the reversed rate of $\mathbf{q}(s_j, a_i, s_k)$ then we can see that RCAT requires that the reversed rate of any active action a has to be constant. By contrast, in our theorem we require the *sum* of the reversed rates of the active transitions to be constant in each state rather that the reversed rate of each active transition. This is a generalization of the condition of RCAT. The distinction presented here is not trivial. In fact, we have shown in [7] different examples in which Theorem 3.2 holds while the original RCAT does not.

4 Conclusion

In this paper we have proposed an extension of the Reversed Compound Agent Theorem (RCAT) for the analysis of product-form models. We have shown that it possible to relax its structural conditions in order to deal with a larger class of product form solutions. As future work is concerned we aim to extend the proof developed in this paper to the ERCAT [4].

References

[1] G. Balbo, S. C. Bruell, and M. Sereno. Product form solution for Generalized Stochastic Petri Nets. *IEEE Trans. on Software Eng.*, 28:915–932, 2002.

[2] P. Harrison and J. Hillston. Exploiting quasi-reversible structures in Markovian process algebra models. *The Computer Journal*, 38(7):510–520, 1995.

[3] P. G. Harrison. Turning back time in Markovian process algebra. *Theoretical Computer Science*, 290(3):1947–1986, January 2003.

[4] P. G. Harrison. Reversed processes, product forms and a non-product form. *Linear Algebra and Its Applications*, 386:359–381, July 2004.

[5] J. Hillston. *A Compositional Approach to Performance Modelling*. PhD thesis, Department of Computer Science, University of Edinburgh, 1994.

[6] J. Hillston and N. Thomas. Product form solution for a class of PEPA models. *Perform. Eval., Elsevier*, 35(3–4):171–192, 1999.

[7] A.. Marin and M. G. Vigliotti A general result for deriving product-form solutions of Markovian models. Submitted, 2009.

[8] M. Sereno. Towards a product form solution for stochastic process algebras. *The Computer Journal*, 38(7):622–632, December 1995.

Using ODEs from PEPA models to derive asymptotic solutions for a class of closed queueing networks

Nigel Thomas
School of Computing Science, Newcastle University, UK
Email: Nigel.Thomas@ncl.ac.uk

***Abstract*—In this paper a class of closed queueing network is modelled in the Markovian process algebra PEPA. It is shown that a fluid flow approximation using ordinary differential equations (ODEs) gives rise to well known asymptotic results. This result gives context to the use of a fluid flow approximation and is potentially useful in cases where the model is not obviously a closed queueing network. The approach is illustrated using examples of a secure key distribution centre and a multi-user query processing system.**

I. Introduction

Since the introduction of stochastic process algebra, there have been many attempts to tackle the state space explosion problem caused by the composition of many parallel components (see [10] for example). One of the more recent approaches to the issue has been the introduction of fluid flow approximations from systems biology to tackle models where there is a large number of instances of a particular component. Such an approach gives rise to a system of ordinary differential equations, which are generally solved by simulation. Following original work by Hillston [11] on biological systems, this style of fluid approximation has also been applied to more traditional computer applications e.g. [5], [17]. However, this approach has met with some scepticism amongst some computer scientists for a number of reasons. The main criticism is that the approximation maps a stochastic model specification on to a deterministic representation, thus the intrinsic randomness of the system is lost. In addition it can be difficult to derive important computer performance measures such as utilisation (because the fluid is always flowing) and interpreting the behaviour of a continuous fragment of a component does not always make sense. For a full description of the application of fluid approximations to PEPA models, see [2], [7], [8], [11].

In this paper we use the ODE approach to derive an analytical solution to a class of model, specified using the Markovian process algebra PEPA [9]. It is shown that this solution is identical to that used for many years as an asymptotic solution to the mean value analysis of closed queueing networks. This has two clear benefits.

- By relating the fluid flow approximation to established results in queueing theory, we gain greater confidence in the use of ODEs as a solution method.
- By deriving the ODE solution directly from the PEPA model specification, the asymptotic results become easily available in the analysis of models which are not obviously closed queueing networks. Thus the applicability of the asymptotic solution is practically extended without the need for specialist knowledge or insight on the part of the modeller.

The paper is organised as follows. In the next section the model and its asymptotic solution are introduced, followed by the PEPA specification of the class of model under investigation. We show how the ODEs can be derived from the PEPA specification and solved analytically to give the asymptotic solution. The process is illustrated by means of two examples; a secure key exchange protocol and a multi-user query processing system. Finally, some conclusions are drawn and potential future work is discussed.

II. The model and its asymptotic solution

Consider a model of a closed queueing network of N jobs circulating around M service stations, denoted 1 to M; each station is either a queueing station or an infinite server station. There are M_q queueing stations. At each queueing station, i, there is an associated queue (bounded at N) operating a FCFS policy and K_i servers which serve jobs at rate r_i. At each infinite server station, j, jobs experience a random delay with mean $1/r_j$. All services are negative exponentially distributed. Let $\mathcal{M} = \{1, 2, \ldots, M\}$ be the set of all queueing stations.

Queueing models of this form have traditionally been solved using mean value analysis [14]. However, this solution becomes costly when N is large and so a number of approximations have been proposed. The simplest amongst these is the asymptotic bound (see Haverkort [6] pp. 245-247).

Define V_i to be the visit count, the ratio of visits made to station i relative to station 1 (hence $V_1 = 1$). Now consider the smallest possible population size, $N = 1$. This solitary job would find each queue empty and experience a delay of $1/r_i$ at each station i at each visit. Hence, the average number of jobs at station i when $N = 1$, $L_i(1)$, is given by the proportion of time a job spends there. Similarly, when $N > 1$ but still small, a job entering a station has a high probability that there will be at least one idle server, hence the delay at station i is still approximately $1/r_i$. The sum of all average queue lengths

must be N, hence,

$$L_i(N) \geq \frac{NV_i}{r_i \sum_{j=1}^{M} \frac{V_j}{r_j}} \quad (1)$$

Now consider the case when N is very large. Assume that there is one queueing station with less service capacity than all the other queueing stations, denoted by $i = 1$. Obviously as $N \to \infty$ the utilisation of this bottleneck station will approach 1 and its throughput will tend to $K_1 r_1$, i.e. it becomes saturated. Obviously, the throughput must balance across all stations, hence, $K_1 r_1 = V_i L_i r_i \ \forall i \geq 2$. Thus,

$$L_i(N) = \frac{K_1 r_1}{V_i r_i} \ , \ i \geq 2 \quad (2)$$

The sum of all queues must equal N, hence

$$L_1(N) \leq N - \sum_{i=2}^{M} \frac{K_1 r_1}{V_i r_i} \quad (3)$$

Thus, for station 1 the approximate average queue length when the population size is N is given by

$$Max \left[\frac{N}{r_1 \sum_{i=1}^{M} \frac{V_i}{r_i}}, N - \sum_{i=2}^{M} \frac{K_1 r_1}{V_i r_i} \right]$$

And for all other stations, i, the approximate average queue length when the population size is N is given by,

$$Max \left[\frac{NV_i}{r_i \sum_{j=1}^{M} \frac{V_j}{r_j}}, \frac{K_1 r_1}{V_i r_i} \right]$$

Clearly a very simple approximation for the average response time has been used to make this calculation for average queue length. However, if our goal is to find the average response time when the population size is N, $W(N)$, a much better estimate can be found by combining this approximation for $L_i(N-1)$ with the mean value analysis estimation to give,

$$W_1(N) = \frac{1}{r_1} + \frac{1}{r_1} Max \left[\frac{N-1}{r_1 \sum_{i=1}^{M} \frac{V_i}{r_i}}, N - 1 - \sum_{i=2}^{M} \frac{K_1 r_1}{V_i r_i} \right] \quad (4)$$

and,

$$W_i(N) = \frac{1}{r_i} + \frac{1}{r_i} Max \left[\frac{(N-1)V_i}{r_i \sum_{j=1}^{M} \frac{V_j}{r_j}}, \frac{K_1 r_1}{V_i r_i} \right] \ , \ 2 \leq i \leq M \quad (5)$$

This is essential a single step of the mean value analysis algorithm (the Nth step) with all previous steps replaced by the asymptotic approximation. The total average respone time, that is the average time from leaving node 1 to subsequently completing another service at node 1, can be found by summing $W_i(N)$ over all nodes, i.

It is a simple matter to consider the case where there is more than one bottleneck, although this is not a concern in this paper. Clearly these two sets of asymptotic results are most accurate at their extremes, i.e. when $N = 1$ or as $N \to \infty$. Thus the asymptotic solution is least accurate when the two curves for L_i meet. There are a number of other approximations and enhancements which seek to improve accuracy and applicability without the additional computational cost associated with mean value analysis.

This form of simple approximation is generally applied across the entire network to derive measures such as system throughput and response time. However, it may also be applied to single nodes as outlined above. In such situations the accuracy of the approximation is greatly variable. The approximation generally works well when queueing only has a significant effect at one station. This situation arises when there is only one queueing station (the remainder being infinite service stations) or where one queueing station has much less service capacity than the others, relative to load.

III. A CLASS OF CLOSED QUEUEING NETWORKS IN PEPA

The model introduced in Section II is now modelled in PEPA. Clearly there are many possible ways to model this system and the particular form of the PEPA model here is a design choice. In particular, the model has been specified in such a way that the ODEs can be derived easily (without further transformation) and all behaviours are named, to improve clarity.

In PEPA a queue station can be modelled as

$$QStation_i \stackrel{def}{=} (service_i, r_i).QStation_i \ , \ \forall i \in \mathcal{M}$$

The infinite server stations are not represented explicitly.

Each job will receive service from a sequence of stations determined by a set of routing probabilities,

$$Job_i \stackrel{def}{=} \sum_{j=1}^{M} (service_i, P_{ij}(i) r_i).Job_j \ , \ 1 \leq i \leq M$$

Where,

$$\sum_{j=1}^{M} P_{ij}(i) = 1 \ , \ 1 \leq i \leq M$$

The entire system can then be represented as follows:

$$\left(\prod_{\forall i \in \mathcal{M}} QStation_i[K_i] \right) \underset{\mathcal{L}}{\bowtie} Job_1[N]$$

Where $\mathcal{L}$ is the set of all action types $service_i$ where $i \in \mathcal{M}$.

The ODEs for such a system are relatively simple to derive directly:

$$\begin{aligned} \frac{d}{dt} Job_i &= \sum_{\forall j \notin \mathcal{M}} P_{ji}(j) r_j Job_j(t) \\ &\quad + \sum_{\forall j \in \mathcal{M}} P_{ji}(j) r_j \ min[K_j, Job_j(t)] \\ &\quad - r_i Job_i(t) \ , \ \forall i \notin \mathcal{M} \end{aligned}$$

$$\begin{aligned} \frac{d}{dt} Job_i &= \sum_{\forall j \notin \mathcal{M}} P_{ji}(j) r_j Job_j(t) \\ &\quad + \sum_{\forall j \in \mathcal{M}} P_{ji}(j) r_j \ min[K_j, Job_j(t)] \\ &\quad - min[K_i, Job_i(t)] r_i \ , \ \forall i \in \mathcal{M} \end{aligned}$$

ODEs such as these can be solved in a number of ways. Most commonly they would be simulated with a suitably small time step to find $Job_i(t)$. In general, this quantity tends to a constant value as $t \to \infty$, i.e. it has a steady state solution.

An alternative method for finding $\lim_{t\to\infty} Job_i(t)$ is to solve the ODEs analytically. Such a solution is based on the assumption that the system of ODEs will eventually reach a steady state. Thus the derivatives will tend to zero as t tends to ∞, i.e.

$$\lim_{t\to\infty} \frac{d}{dt} Job_i \to 0 \ , \ 1 \leq i \leq M$$

This gives rise to the following set of simple simultaneous equations:

$$\begin{aligned}
&\lim_{t\to\infty} \sum_{\forall j \notin \mathcal{M}} P_{ji}(j) r_j Job_j(t) \\
&+ \sum_{\forall j \in \mathcal{M}} P_{ji}(j) r_j \ min[K_j, Job_j(t)] \\
&= \lim_{t\to\infty} r_i Job_i(t) \ , \ \forall i \notin \mathcal{M}
\end{aligned}$$

$$\begin{aligned}
&\lim_{t\to\infty} \sum_{\forall j \notin \mathcal{M}} P_{ji}(j) r_j Job_j(t) \\
&+ \sum_{\forall j \in \mathcal{M}} P_{ji}(j) r_j \ min[K_j, Job_j(t)] \\
&= \lim_{t\to\infty} r_i \ min[K_i, Job_i(t)] \ , \ \forall i \in \mathcal{M}
\end{aligned}$$

There are M equations, but each equation can be expanded (with respect to min function) in up to 2^{M_q} ways.

Define the probability that a component will evolve from Job_i to Job_j, without revisiting Job_i, as follows:

$$P_{ij}(i) = p_{ij} + \sum_{\forall k \notin \sigma} p_{ik} P_{kj}(i)$$

Clearly the system is irreducible if

$$P_{ij}(i) > 0 \ \forall i, j \ , \ i \neq j$$

Define L_i to be the steady state average number of components behaving as Job_i, given by

$$L_i = \lim_{t\to\infty} Job_i$$

Now select $i \in \mathcal{M}$ such that[1]

$$P_{ij}(i) r_i K_i < P_{ji}(j) r_j K_j \ \forall j \in \mathcal{M} \ , \ j \neq i \qquad (6)$$

Hence,

- If $K_i \leq L_i$ then

$$P_{ij}(i) r_i K_i = P_{ji}(j) r_j L_j \ , \ \forall j \qquad (7)$$

- If $K_i > L_i$ then

$$P_{ij}(i) r_i Job_i = P_{ji}(j) r_j L_j \ , \ \forall j \qquad (8)$$

[1]If (6) does not hold then there is no unique solution for this fluid system except when N is very small. Instead the value of L_i will depend on the initial values $Job_i(0)$ $\forall i$.

Thus, if $K_i \leq L_i$ then

$$\begin{aligned}
L_j &= \frac{P_{ij}(i) r_i K_i}{P_{ji}(j) r_j} \ , \ \forall j \neq i & (9) \\
L_i &= N - \sum_{\forall j \neq i} L_j = N - \sum_{\forall j \neq i} \frac{P_{ij}(i) r_i K_i}{P_{ji}(j) r_j} & (10)
\end{aligned}$$

Otherwise, if $K_i \geq L_i$ then

$$\begin{aligned}
L_j &= \frac{P_{ij}(i) r_i}{P_{ji}(j) r_j} L_i \ , \ \forall j \neq i & (11) \\
L_i &= N - \sum_{\forall j \neq i} L_j = \frac{N}{1 + \sum_{\forall j \neq i} \frac{P_{ij}(i) r_i}{P_{ji}(j) r_j}} & (12)
\end{aligned}$$

Clearly (10) and (12) meet when $K_i = L_i$. This point is given by the population size $N = N^*$, given by

$$N^* = K_i \left(1 + \sum_{\forall j \neq i} \frac{P_{ij}(i) r_i}{P_{ji}(j) r_j} \right)$$

Thus, if $N \leq N^*$ then

$$\begin{aligned}
L_j &= \frac{P_{ij}(i) r_i}{P_{ji}(j) r_j} L_i \ , \ \forall j \neq i & (13) \\
L_i &= N - \sum_{\forall j \neq i} L_j = \frac{N}{1 + \sum_{\forall j \neq i} \frac{P_{ij}(i) r_i}{P_{ji}(j) r_j}} & (14)
\end{aligned}$$

Otherwise, if $N \geq N^*$ then

$$\begin{aligned}
L_j &= \frac{P_{ij}(i) r_i K_i}{P_{ji}(j) r_j} \ , \ \forall j \neq i & (15) \\
L_i &= N - \sum_{\forall j \neq i} L_j = N - r_i K_i \sum_{\forall j \neq i} \frac{P_{ij}(i)}{P_{ji}(j) r_j} & (16)
\end{aligned}$$

Clearly, the visit count is given by $V_i = P_{ij}(i)/P_{ji}(j)$. Hence (13), (14), (15) and (16), are equivalent to (1), (2) and (3). Observe also that (13) and (14) hold, with arbitrary i, regardless of the existence of (6) as long as $L_j \leq K_j \ \forall j$.

From these expressions for L_j we can derive the average response time at station j when the population size is N, $W_j(N)$ in the same was as 4 and 5.

$$\begin{aligned}
W_j(N) &= \frac{1}{r_j} \ , \ L_j(N-1) + 1 \leq K_j \\
W_j(N) &= \frac{L_j(N-1) + 1}{K_j r_j} \ , \ L_j(N-1) + 1 > K_j
\end{aligned}$$

Where $L_j(N)$ is the average number of jobs at station j when the population size is N. This computation for $W_j(N)$ is based on the queueing theory result of an arrival as random observer, see Mitrani [12] page 141 for example. If the random observer sees a free server, then the average response time will be the average service time. However, if the random observer sees all the servers busy, then the average response time will be the average service time plus the time it takes for one server to become available (including scheduling the other jobs waiting ahead of the random observer).

In addition we can derive an expression for utilisation at the bottleneck queueing station i, U_i, based on the flow into the station being equal to the available service.

$$U_i = \sum_{\forall j \neq i} \frac{P_{ji}(j) L_j}{r_j K_i r_i}$$

IV. Example 1: A Secure Key Distribution Centre

Consider a model of the classic Needham-Schroeder key distribution protocol (taken from [19]) specified as follows:

$$\begin{aligned}
KDC &\stackrel{def}{=} (response, r_p).KDC \\
\\
Alice_0 &\stackrel{def}{=} (request, r_q).Alice_1 \\
Alice_1 &\stackrel{def}{=} (response, r_p).Alice_2 \\
Alice_2 &\stackrel{def}{=} (sendBob, r_B).Alice_3 \\
Alice_3 &\stackrel{def}{=} (sendAlice, r_A).Alice_4 \\
Alice_4 &\stackrel{def}{=} (confirm, r_c).Alice_5 \\
Alice_5 &\stackrel{def}{=} (usekey, r_u).Alice_0
\end{aligned}$$

The system is then defined as:

$$KDC[K] \underset{response}{\bowtie} Alice_0[N]$$

Where, K is the number of *KDC*'s and N is the number of client pairs (*Alices*'s).

It is a simple matter to write down the ODEs for this system as follows.

$$\begin{aligned}
\frac{d}{dt}Alice_0 &= r_u Alice_5(t) - r_q Alice_0(t) \\
\frac{d}{dt}Alice_1 &= r_q Alice_0(t) - r_p\ min(KDC(t), Alice_1(t)) \\
\frac{d}{dt}Alice_2 &= r_p\ min(KDC(t), Alice_1(t)) - r_B Alice_2(t) \\
\frac{d}{dt}Alice_3 &= r_B Alice_2(t) - r_A Alice_3(t) \\
\frac{d}{dt}Alice_4 &= r_A Alice_3(t) - r_c Alice_4(t) \\
\frac{d}{dt}Alice_5 &= r_c Alice_4(t) - r_u Alice_5(t) \\
\frac{d}{dt}KDC &= 0
\end{aligned}$$

In this analysis we are interested primarily in the number of client pairs awaiting a response from the *KDC* (or *KDC*'s) from a population of size N, which we denote as $L(N)$ This is represented in the model by the number of $Alice_1$'s; $L(N) = \lim_{t\to\infty} Alice_1(t)$ when there are N client pairs (*Alice*'s) in the population.

If the system reaches a steady state then all the derivatives will tend to zero as t tends to ∞, i.e.

$$\lim_{t\to\infty} \frac{d}{dt} Alice_i \to 0\ ,\ 0 \leq i \leq 5$$

Hence,

$$\begin{aligned}
\lim_{t\to\infty} r_p\ min(KDC(t), Alice_1(t)) &= \lim_{t\to\infty} r_B Alice_2(t) \\
&= \lim_{t\to\infty} r_A Alice_3(t) \\
&= \lim_{t\to\infty} r_c Alice_4(t) \\
&= \lim_{t\to\infty} r_u Alice_5(t) \\
&= \lim_{t\to\infty} r_q Alice_0(t)
\end{aligned}$$

Thus we only need to solve this set of simple parallel equations to find $L(N)$. If $KDC(t) \geq Alice_1(t)$ then the ODEs give rise to

$$L(N) = \lim_{t\to\infty} Alice_1 = \frac{N r_x}{r_x + r_p} \tag{17}$$

If $KDC(t) \leq Alice_1(t)$ then the ODEs give rise to

$$L(N) = \lim_{t\to\infty} Alice_1 = \frac{N r_x - K r_p}{r_x} \tag{18}$$

Where r_x is given by

$$r_x = \left(\frac{1}{r_q} + \frac{1}{r_B} + \frac{1}{r_A} + \frac{1}{r_c} + \frac{1}{r_u} \right)^{-1} \tag{19}$$

(17) and (18) meet when $KDC(t) = Alice_1(t)$ for a given population size N^*, hence, with (19) we get,

$$N^* = K + \frac{K r_p}{r_x} = K + K r_p \left(\frac{1}{r_q} + \frac{1}{r_B} + \frac{1}{r_A} + \frac{1}{r_c} + \frac{1}{r_u} \right)$$

Figure 1 shows the average response time of the *KDC*, found approximately using (17) and (18) and computed exactly using mean value analysis [16]. Clearly, when the service rate is smaller, the response time is larger and its rate of increase is larger. As noted above, there is a difference between the two solutions around N^*, which is clearly evident.

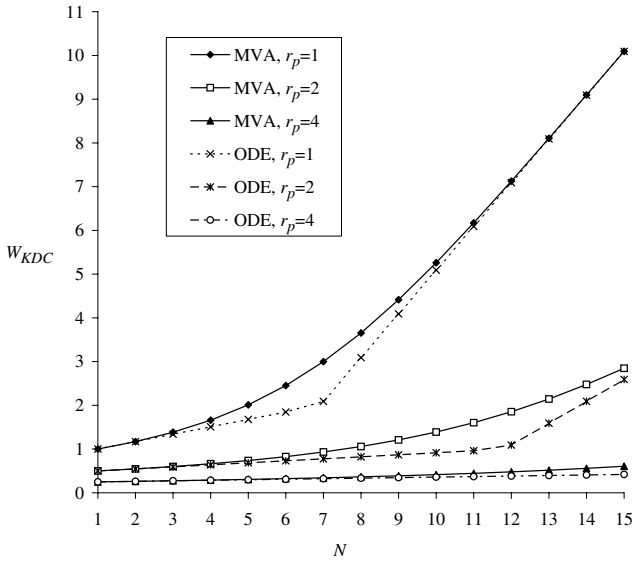


Fig. 1. Average response time at the *KDC* varied with population size ($r_q = r_B = r_A = r_c = 1$, $r_u = 1.1$, $K = 1$)

Figure 2 shows the average queue length at the *KDC*, L_{KDC} for this system when there is either one fast server or K slower servers. When the population size is large ($N > 30$ in this case) the *KDC* becomes saturated and there is consequently no difference in the service rate offered between the two cases shown. However, when N is smaller, there will be periods where one or more of the K servers will be idle, thus reducing the overall service capacity offered. Hence, for smaller N, a single fast server will out perform multiple slower servers with the same overall capacity. Once again, there is a clear divergence around N^*.

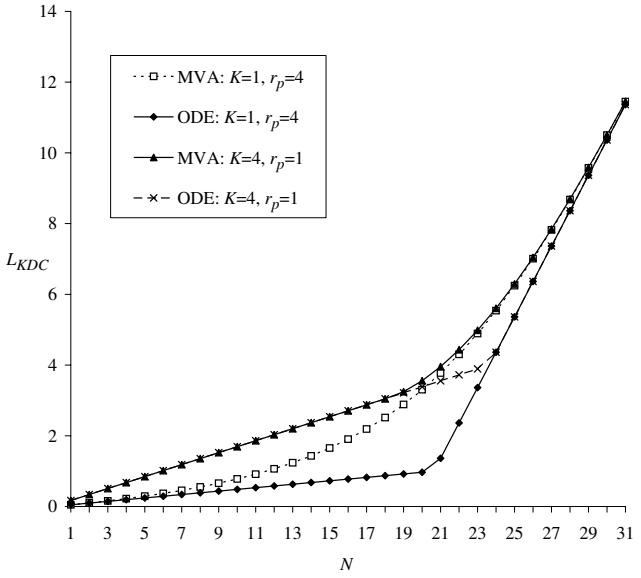


Fig. 2. Average queue length at *KDC* varied with population size ($r_q = r_B = r_A = r_c = 1$, $r_u = 1.1$)

V. Example 2: A Multi-User Query Processing System

Consider the following PEPA specification of a classic model taken from Lazowska et al [13].

$$\begin{aligned} Proc &\stackrel{def}{=} (service, \mu).Proc \\ Disk &\stackrel{def}{=} (write, \eta).Disk \end{aligned}$$

$$\begin{aligned} User_1 &\stackrel{def}{=} (think, \xi).User_2 \\ User_2 &\stackrel{def}{=} (service, p\mu).User_1 \\ &\quad +(service, (1-p)\mu).User_3 \\ User_3 &\stackrel{def}{=} (write, \eta).User_1 \end{aligned}$$

The entire system is then specified as

$$(Proc||Disk[K]) \underset{\{service,\ write\}}{\bowtie} User_1[N]$$

This system depicts a processor and an array of K independent disks. Users request a service from the processor. After this they either think for a while, before making another request, or their result requires writing to a disk before thinking and then another request.

The ODEs are given as

$$\begin{aligned} \frac{d}{dt}User_1(t) &= p\mu\ min[1, User_2(t)] \\ &\quad +\eta\ min[K, User_3(t)] \\ &\quad -\xi User_1(t) \\ \frac{d}{dt}User_2(t) &= \xi User_1(t) - \mu\ min[1, User_2] \\ \frac{d}{dt}User_3(t) &= (1-p)\mu\ min[1, User_2(t)] \\ &\quad -\eta\ min[K, User_3(t)] \end{aligned}$$

If the system reaches a steady state then all the derivatives will tend to zero as t tends to ∞, i.e.

$$\lim_{t \longrightarrow \infty} \frac{d}{dt} User_i \longrightarrow 0\ ,\ 0 \leq i \leq 5$$

Define $L_i = \lim_{t \longrightarrow \infty} User_i$ to be the steady state average number of users at each point in the system. Hence,

$$\begin{aligned} p\mu\ min[1, L_2] + \eta\ min[K, L_3] &= \xi L_1 \\ \xi L_1 &= \mu\ min[1, L_2] \\ (1-p)\mu\ min[1, L_2] &= \eta\ min[K, L_3] \end{aligned}$$

There are two possible bottlenecks in this system If $(1-p)\mu < K\eta$ then the bottleneck is the processor.

- If $1 \leq L_2$ then

$$\begin{aligned} L_1 &= \frac{\mu}{\xi} \\ L_3 &= \frac{(1-p)\mu}{\eta} \\ L_2 &= N - \frac{\mu}{\xi} - \frac{(1-p)\mu}{\eta} \end{aligned}$$

- If $1 \geq L_2$ then

$$\begin{aligned} L_1 &= \frac{\mu}{\xi}L_2 \\ L_3 &= \frac{(1-p)\mu}{\eta}L_2 \\ L_2 &= N - \frac{\mu}{\xi}L_2 - \frac{(1-p)\mu}{\eta}L_2 \\ &= \frac{N\xi\eta}{\xi\eta + \mu\eta + \mu\xi(1-p)} \end{aligned}$$

In this case,

$$N^* = \frac{\xi\eta + \mu\eta + \mu\xi(1-p)}{\xi\eta}$$

Alternatively, if $(1-p)\mu > K\eta$ then the bottleneck is writing to the disks.

- If $K \leq L_3$ then

$$\begin{aligned} L_1 &= \frac{\eta K}{(1-p)\xi} \\ L_2 &= \frac{\eta K}{(1-p)\mu} \\ L_3 &= N - \frac{\eta K}{\xi(1-p)} - \frac{\eta K}{\mu(1-p)} \end{aligned}$$

- If $K \geq L_3$ then

$$\begin{aligned} L_1 &= \frac{\eta}{(1-p)\xi} L_3 \\ L_2 &= \frac{\eta}{(1-p)\mu} L_3 \\ L_3 &= N - \frac{\eta}{(1-p)\xi} L_3 - \frac{\eta}{(1-p)\mu} L_3 \\ &= \frac{N\xi\mu(1-p)}{\xi\mu(1-p) + \eta\mu + \eta\xi} \end{aligned}$$

In this case,

$$N^* = K + \frac{\eta K}{\xi(1-p)} + \frac{\eta K}{\mu(1-p)}$$

If $(1-p)\mu = K\eta$ then the solution will depend on the initial values of $User_1(0)$, $User_2(0)$ and $User_3(0)$, unless,

$$\frac{N\xi\eta}{\xi\eta + \mu\eta + \mu\xi} \leq 1$$

and,

$$\frac{N\xi\mu}{\xi\mu + K\eta\mu + K\eta\xi} \leq K$$

In which case L_i is given by

$$\begin{aligned} L_1 &= \frac{N\mu\eta}{\xi\eta + \mu\eta + \mu\xi} \\ L_2 &= \frac{N\xi\eta}{\xi\eta + \mu\eta + \mu\xi} \\ L_3 &= \frac{N(1-p)\mu\xi}{\xi\eta + \mu\eta + \mu\xi} \end{aligned}$$

Figures 3 and 4 show the average queue lengths at the processor and the disk array for various values of p, where the processor is the bottleneck (Figure 3) and where the disk array is the bottleneck (Figure 4). In both cases results are shown as calculated by the ODE method in this paper and the mean value analysis method from [16].

It can be seen that when p is relatively large, the approximation works well (except around N^*). Whereas when p is smaller, particularly when p is close to 0.5, it is much poorer, and even diverging with N. It might perhaps be surprising that the ODE and MVA results are not closer when $p = 0.1$. After all, in this scenario, most jobs will visit the disk array and experience a long delay there. However, even when $p = 0.1$ queueing effects still have an effect at the processor and this causes a difference between the two methods.

Clearly, the accuracy of the ODE approximation of average queue length is sensitive to p. However, as stated earlier, the asymptotic solution is generally applied across the entire network, and not at an individual station. Therefore it is interesting to observe the accuracy of system wide metrics. Figure 5 shows the average response time for the entire system, computed as

$$W = \sum_{i=1}^{M} V_i W_i$$

Where V_i is the visit count and $min[V_1, \ldots, V_M] = 1$.

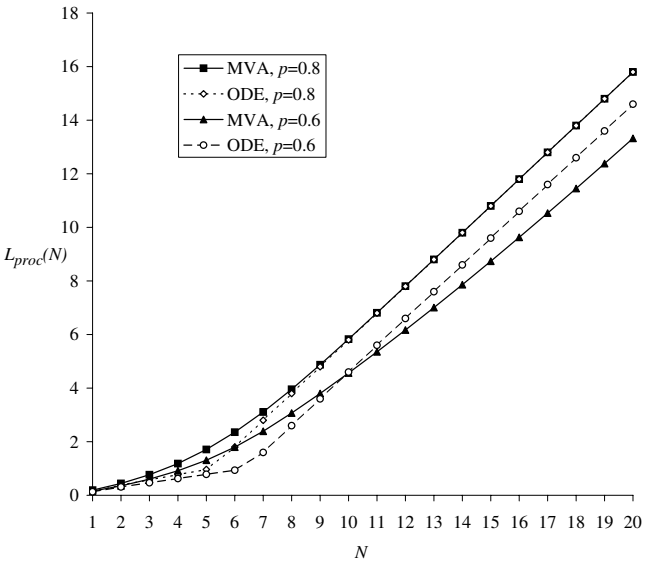


Fig. 3. Average queue length at processor and disk array varied with population size ($\xi = 10\mu = 30, \eta = 5,\ K = 3$)

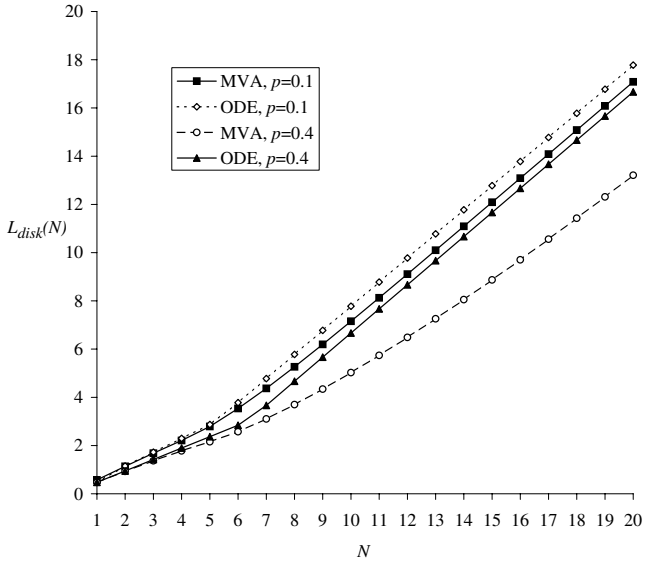


Fig. 4. Average queue length at processor and disk array varied with population size ($\xi = 10\mu = 30, \eta = 5,\ K = 3$)

Clearly the system response time is much less sensitive to the errors in the average number of jobs in each queue than we might naively expect. Indeed, Figure 5 shows only a very small divergence between MVA and ODE calculations, even when $p = 0.4$ (the worst case in the earlier graphs). The explanation for this is relatively simple, in that the maximum error in predicting the queue lengths is caused when the service capacity at each queueing station are relatively similar. Hence, when computing the average response time, we replace a delay at one station with a very similar delay at the other. As such, the errors, to an extent, disappear when aggregated across the

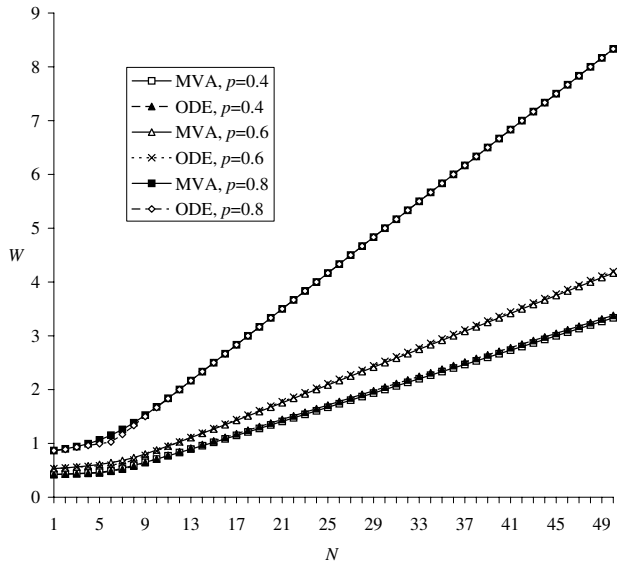


Fig. 5. Average system response time varied with population size ($\xi = 10\mu = 30, \eta = 5$, $K = 3$)

whole network in this way.

VI. Conclusions and further work

It has been demonstrated that the fluid approximation for a class of PEPA models coincides with the well known asymptotic approximation for a corresponding class of closed queueing network. This result is potentially useful as an alternative means for characterising models of queueing networks specified using PEPA, particularly when the model specified is not obviously a queueing model. This is the first class of fluid PEPA model for which there is an explicit expression for where the fluid solution is least accurate with respect to the exact solution of the stochastic model. The derivation of ODEs used in this paper is based on the work of Hillston [11] and is incorporated into the PEPA Eclipse Plug-in [18], facilitating easy numerical solution.

The result in this paper is limited to a class of cyclic queueing model where each station can perform just one action type. However, the asymptotic approximation applies to a much wider class of model. Thus it should be possible to extend this result to consider PEPA models with multiple competing action types at each Job_i derivative, each occurring at different rates but with the overall rate capped by a $QStation_j$ component. In addition, the asymptotic result holds for general service distributions, suggesting that the applicability of the fluid approximation in PEPA potentially extends beyond its conventional Markovian semantics. These investigations are left as ongoing work.

The result here is also limited to the case where there is a single bottleneck, unless the population is small enough that $L_i \leq K_i \ \forall i$. It is clearly possible to easily compute the average queue length for systems where there is more than one bottleneck, however it is not possible to find a unique solution directly from the ODEs, which is the aim of this paper. Furthermore, the approximation is shown to be accurate only when there are significant queueing effects at one station only. This gives some further insight as to the kind of model where ODE analysis is (in)appropriate.

Acknowledgements

The author is indebted to A. Clark, A. Duguid, S. Gilmore and M. Tribastone of the University of Edinburgh for invaluable comments on earlier work which contributed to this paper, in particular for clarifying aspects of the PEPA Eclipse Plug-in and the apparent rate. The example of the Key Distribution Centre is based on earlier work by Zhao and Thomas [19].

References

[1] J. Bradley, S. Gilmore and N. Thomas, Performance analysis of Stochastic Process Algebra models using Stochastic Simulation, in: *Proceedings of 20th IEEE International Parallel and Distributed Processing Symposium*, IEEE Computer Society, 2006.

[2] J. Bradley, A ticking clock: Performance analysis of a Circadian rhythm with stochastic process algebra, in: C. Juiz and N. Thomas (eds.), *Computer Performance Evaluation: 5th European Performance Engineering Workshop*, LNCS 5261, Springer Verlag, 2008.

[3] A. Clark, A. Duguid, S. Gilmore and M. Tribastone, Partial evaluation of PEPA models for fluid-flow analysis, in: *Computer Performance Engineering: Proceedings of the 5th European Workshop on Performance Engineering (EPEW)*, LNCS 5261, Springer-Verlag, 2008.

[4] G. Clark and S. Gilmore and J. Hillston and N. Thomas, *Experiences with the PEPA Performance Modelling Tools*, IEE Proceedings - Software, pp. 11-19, 146(1), 1999.

[5] A. Duguid, Coping with the Parallelism of BitTorrent: Conversion of PEPA to ODEs in dealing with State Space Explosion. in: E. Asarin and P. Bouyer (eds.), *Formal Modeling and Analysis of Timed Systems*, LNCS 4202, Springer-Verlag, 2006.

[6] B. Haverkort, *Performance of Computer Communication Systems: A model Based Approach*, Wiley, 1998.

[7] Richard Hayden, Addressing the state space explosion problem for PEPA models through fluid-flow approximation, Undergraduate Project Dissertation, Imperial College London, 2007.

[8] R. Hayden and J. Bradley, Fluid-flow solutions in PEPA to the state space explosion problem, ValueTools 2008.

[9] J. Hillston, *A Compositional Approach to Performance Modelling*, Cambridge University Press, 1996.

[10] J. Hillston, Exploiting Structure in Solution: Decomposing Compositional Models, in: E. Brinksma *et al*, *Lectures on Formal Methods and Performance Analysis*, LNCS 2090, Springer-Verlag, 2003.

[11] J. Hillston, Fluid flow approximation of PEPA models, in: *Proceedings of QEST'05*, pp. 33-43, IEEE Computer Society, 2005.

[12] I. Mitrani, *Probabilistic Modelling*, Cambridge University Press, 1998.

[13] E. Lazowska, J. Zahorjan, S. Graham and K. Sevcik, *Quantitative System Performance*, Prentice-Hall, 1984.

[14] M. Reiser and S. Lavenberg, Mean value analysis of closed multichain queueing networks, *JACM*, **22**(4), pp. 313-322, 1980.

[15] W. Stallings, *Cryptography and Network Security: Principles and Practice*, Prentice Hall, 1999.

[16] N. Thomas and Y. Zhao, Mean value analysis for a class of PEPA models, in: Proceedings of 6th European Performance Engineering Workshop, LNCS 5652, Springer-Verlag, 2009.

[17] N. Thomas and Y. Zhao, Fluid flow analysis of a model of a secure key distribution centre, in: *Proceedings 24th Annual UK Performance Engineering Workshop*, Imperial College London, 2008.

[18] M. Tribastone, The PEPA plug-in project, in: *Proceedings of 4th International Conference on the Quantitative Evaluation of Systems (QEST)*, pp. 53-54, IEEE Computer Society, 2007.

[19] Y. Zhao and N. Thomas, Approximate solution of a PEPA model of a key distribution centre, in: *Performance Evaluation - Metrics, Models and Benchmarks: SPEC International Performance Evaluation Workshop*, pp. 44-57, LNCS 5119, Springer-Verlag, 2008.

Response-time Profiles for PEPA models compiled to ODEs

Allan Clark*

Abstract

This paper reports on a new method for calculating response-time profiles from PEPA models. This method is particularly suitable for models which are too large for the underlying continuous-time Markov chain to be computed and are thus translated into a system of ordinary differential equations. A response-time profile examines the probability that a particular component receives a corresponding 'response' a given time after it made a 'request'. Any two source and target activities will work provided that both are initiated or observed by the same component kind. More general passage-time profiles analysing the probability of moving from any set of source states to any set of target states are not considered here.

This is a draft of a more complete paper to follow.

1 Introduction

Stochastic process algebras are used to build models describing real-world systems. These models are then analysed giving performance results which we hope are applicable to the real-world system. The real-world system may not yet exist or we may be considering changing the real-world system or its environment may chnge outside of our control. Therefore we are using modelling to predict the performance of a combination of system and environment without the cost of deploying a real-world test, which may not even by feasible and is likely prohibitively expensive.

Such models whether described in a process algebra such as PEPA[1] or a graphical notation such as Petri Nets are often translated into a continuous-time Markov chain (CTMC). However this approach to evaluating the model suffers from the well-known problem of state-space explosion. As we increase the populations of the components involved in the model — for example increasing the number of *user* or *client* components — the size of the resulting state-space grows exponentially. Even utilising techniques to dampen the effect of state-space explosion, such as aggregation[2, 3], this greatly limits the applicability of this approach to analysing a model, to the extent that in recent years modellers have often turned to other techniques to analyse a model which do not require the generation of the entire state-space.

In the world of PEPA the break through for this style of analysis was Jane Hillston's 2005 paper[4] which describes the automatic translation of a PEPA model into a set of coupled ordinary differential equations (ODEs). The number

*LFCS, University of Edinburgh, a.d.clark@ed.ac.uk

of equations produced from the translation depends on the number of distinct component states and not the populations of the component types. This means that we are able to analyse models that would result in state-space sizes of the order of 10^{1000} and beyond. Since then, Stochastic Simulation Algorithm(SSA)[5] has also been used[6] to analyse large-scale models.

The advantage of using such techniques is clear, models which were previously infeasible to analyse (using CTMCs) now fall within the realm of models appropriate for analysis via description in PEPA. The disadvantage is the loss of the vast bank of knowledge of analysis techniques for continuous-time Markov chains. One kind of analysis which can be achieved in various ways including uniformisation[7, 8, 9] (also known as randomisation), is the extraction of *passage-time* quantiles. This allows the calculation of the probability of moving from one set of source states to another set of target states at or within a given time after a source state is entered. We call this a *response-time profile* when measuring the probability of completing a passage between two events observable by a single component. This is often a single component actively initiating a request activity and then waiting to passively or actively cooperate in a response activity. Although we call this a response-time profile the initiating and completing activities can conceptually have nothing to do with a request and response, however for the remainder of this paper we will refer to them as request and response activities. A response-time profile is a more detailed report of the responsiveness of a system than an average response-time.

For large-scale systems which cannot be compiled to a CTMC we have been limited to the computation of average response-times. Simulation of the derived system of ODEs is used to predict the evolution of the population of each component type over time within the system. Where this converges to an unchanging system in which the population of each component type remains the same we can classify this as the steady-state of the system. Using this steady-state and an application of *Little's Law*[10] we can extract average response-times[11]. The main contribution of this paper is the generalisation of this technique into one for obtaining a full response-time profile computing the probability of a component observing the completion of a response at a given time after the initiation of the request.

2 Response-time Profiles

In this section we detail how to obtain a complete response-time profile from a PEPA model which we compile into a set of ODEs. We build upon the general idea originally proposed by Bradley et al in [12]. We show that this will create an incorrect response-time profile for most models and how we can rectify this.

2.1 Absorbing Probes

The essential idea is to 'seed' a model in which all of the clients are in their single 'source' state. Whereby 'seed' means to perform a transient analysis beginning with a specified set of component populations. The source state is the start of the response-passage for which we are interested in computing the response-time profile. This is a state to which the client component transitions to when performing the initiating *request* action. For the time being we are

$$
\begin{array}{lcl}
((\alpha,\lambda).P) \rhd_{(U)} H & = & \begin{cases} (\alpha,\lambda).\mathbf{Stop} & : \quad P \in H \\ (\alpha,\lambda).(P \rhd_{(U\cup\{P\})} H) & : \quad P \notin H, P \notin U \\ (\alpha,\lambda).P & : \quad P \notin H, P \in U \end{cases} \\
(P + Q) \rhd_{(U)} H & = & (P \rhd_{(U)} H) + (Q \rhd_{(U)} H) \\
(P \backslash L) \rhd_{(U)} H & = & (P \rhd_{(U)} H) \backslash L \\
(P \underset{L}{\bowtie} Q) \rhd_{(U)} H & = & (P \rhd_{(U)} H) \underset{L}{\bowtie} (Q \rhd_{(U)} H)
\end{array}
$$

Figure 1: The $\rhd_U$ operator is defined to convert the component states in the set H into absorbing states. The component states in the set U are those which have already been visited.

restricting ourselves to models in which there is only one source state for each client component.

Having set the initial population of the model such that all clients are in the source state we then obtain a time-series analysis of the model until all clients have completed one response-passage, that is until all of them have performed a *response* action. However the problem with this approach is that the clients which have completed their response-passage will, after some delay, re-enter the source state and subsequently states along the response-passage. Therefore there is no way to tell how many have completed at least one response-passage. In their paper cited above, Bradley et al solve this problem by making all the client processes absorbing upon completion of a passage. This is done using an absorbing operator shown in Figure 1.

An alternative to defining a new operator over PEPA models is to alter the way in which the passage specification probe is translated. A stochastic probe[13], is a sequential PEPA component which is attached to some portion of the model in order to make observations on that portion of the model simpler. The probe component never initiates any actions but instead only observes, through passive cooperation, the activities performed by the portion of the model to which it is attached. Probe components can be specified in a succinct language and translated into sequential PEPA components automatically.

Generally a stochastic probe specification is translated into a cyclic sequential PEPA component. Instead it is possible to translate the probe specification in to a deadlocking component, which upon observing the target activity moves into a state from which there are no outward transitions. In addition we must be careful that the alphabet of the probe — that is the set of actions observed by the probe — is the same as the alphabet of the client component, thus ensuring that the client cannot freely perform an action without the probe cooperating. In this way we ensure that once the probe is in the absorbing state and cannot perform/observe/cooperate any action then the client component is also blocked.

Note that this is a departure from the usual probe usage which must certify that the addition of the probe does not alter the behaviour of the model, however since altering the model is exactly what we wish to achieve here that is to be expected. The following is an example model of a web commerce site and a translated probe specification which measures from a client's first *browse* until their eventual $getConfirm$ which concludes their session.

$$
\begin{array}{lcl}
User & \stackrel{def}{=} & (browse, r_{browse}).Browsing \\
 & + & (buy, r_{buy}).Buying \\
Browsing & \stackrel{def}{=} & (getPage, r_{getPage}).User \\
Buying & \stackrel{def}{=} & (getConfirm, r_{getConfirm}).User
\end{array}
$$

$$
\begin{array}{lcl}
Server & \stackrel{def}{=} & (getPage, r_{sendPage}).Server \\
 & + & (getConfirm, r_{sendConfirm}).Server
\end{array}
$$

$(Server[M]) \underset{\mathcal{L}}{\bowtie} (User[N])$
where $\mathcal{L} = \{getPage, getConfirm\}$

$$
\begin{array}{lcl}
Probe & \stackrel{def}{=} & (browse, \top).Running \\
 & + & (getConfirm, \top).Probe \\
 & + & (buy, \top).Probe \\
 & + & (getPage, \top).Probe \\
Running & \stackrel{def}{=} & (getConfirm, \top).Done \\
 & + & (browse, \top).Running \\
 & + & (buy, \top).Running \\
 & + & (getPage, \top).Running \\
Done & \stackrel{def}{=} & \textbf{Stop}
\end{array}
$$

This component is attached to the user component and the cooperation $(User \underset{*}{\bowtie} Probe)$ is partially evaluated to the following PEPA component which can then be multiplied via the array operator and used in our example model:

$$
\begin{array}{lcl}
ProbeUser & \stackrel{def}{=} & (browse, r_{browse}).RunningBrowsing \\
 & + & (buy, r_{buy}).ProbeBuying \\
ProbeBuying & \stackrel{def}{=} & (getConfirm, r_{getConfirm}).ProbeUser \\
RunningBrowsing & \stackrel{def}{=} & (getPage, r_{getPage}).RunningUser \\
RunningUser & \stackrel{def}{=} & (browse, r_{browse}).RunningBrowsing \\
 & + & (buy, r_{buy}).RunningBuying \\
RunningBuying & \stackrel{def}{=} & (getConfirm, r_{getConfirm}).\underline{Done} \\
Done & \stackrel{def}{=} & \textbf{Stop}
\end{array}
$$

The resulting model is translated into a set of ODEs, the initial population for the $UserProbe$ component is set such that the entire population is in the source state $RunningBrowsing$ (such that, in this case, the states $ProbeUser$ and $ProbeBuying$ are unreachable).

From this the CDF of the response-passage at time t is $\frac{Done(t)}{RunningBrowsing(0)}$ or rather the number of clients which have completed the passage by time t divided by the total population of clients. This is shown in Figure 2.

Since we are also able to calculate the average response-time of this same passage, we can show that the CDF computed in this way is overly optimistic. Shown on the diagram are two areas, A and B. Area A is the area underneath the CDF from 0 until the average response-time. Area B is the area above the CDF from the average response-time to infinity. If the CDF is correctly calculated then these two areas should be equal, whereas on the diagram clearly they are not equal.

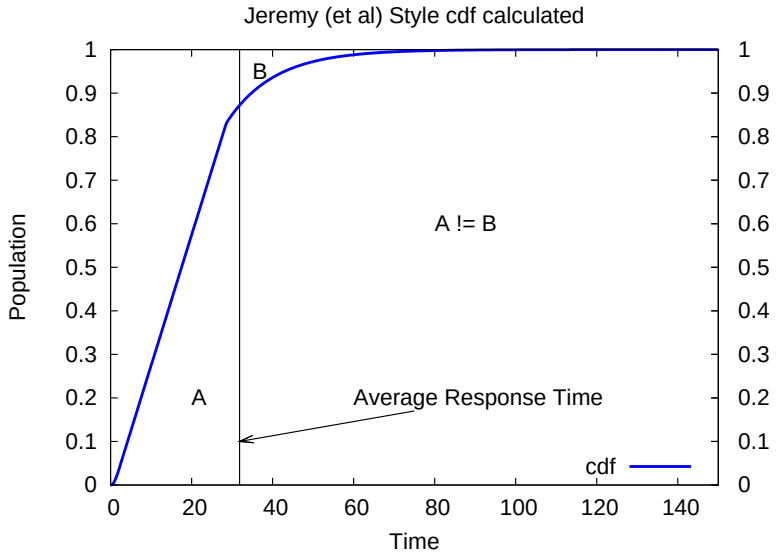


Figure 2: A graph of the CDF for the example response-passage as computed by the Bradley et al method using ODEs.

2.2 One-Run Probes

The main reason that the above method for calculating the CDF is optimistic is that once the individual clients have completed their own response-passage they are in a blocked state and can therefore not re-compete for the same resources. The later finishing clients therefore have artificially exclusive use of the shared resources.

A further problem, which may instead cause the computed CDF to be pessimistic, is that this method makes no use of when a request action is likely to occur.

To tackle our first problem we introduce the notion of a *one-run* probe. Figure 3 shows a one-run probe together with a typical probe and the above described absorbing probe. The key point is that once the one-run probe transitions into its absorbing state it will never transition from it, but because it allows all of its probe alphabet in cooperation with the user component, the user component is not blocked from behaving as a non-probed user component. However because it is in cooperation with a finished probe we can tell how many such user components there currently are. That is we can tell how many user components have completed one response-passage even though they are now currently acting as a non-probed user component and in particular offering up competition for the shared resources.

Once we have added the one-run probe to our example model and partially evaluated the resulting cooperation between probe and user components we obtain the following PEPA model:

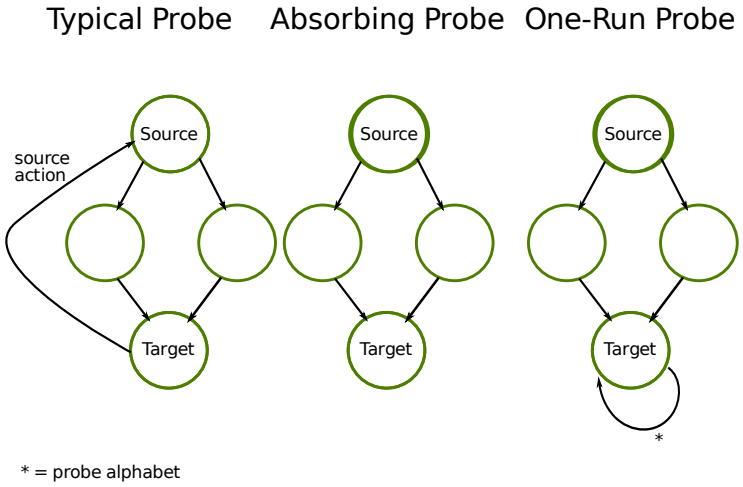


Figure 3: A typical probe together with an absorbing and a one-run probe

$$
\begin{array}{lcl}
RunningBrowsing & \stackrel{def}{=} & (getPage, r_{getPage}).RunningUser \\
RunningUser & \stackrel{def}{=} & (browse, r_{browse}).RunningBrowsing \\
 & + & (buy, r_{buy}).RunningBuying \\
RunningBuying & \stackrel{def}{=} & (getConfirm, r_{getConfirm}).RanUser \\
RanUser & \stackrel{def}{=} & (browse, r_{browse}).RanBrowsing \\
 & + & (buy, r_{buy}).RanBuying \\
RanBrowsing & \stackrel{def}{=} & (getPage, r_{getPage}).RanUser \\
RanBuying & \stackrel{def}{=} & (getConfirm, r_{getConfirm}).RanUser \\
 & & \\
Server & \stackrel{def}{=} & (getPage, r_{sendPage}).Server \\
 & + & (getConfirm, r_{sendConfirm}).Server
\end{array}
$$

$(Server[M]) \underset{\mathcal{L}}{\bowtie} (RunningBrowsing[N])$
where $\mathcal{L} = \{getPage, getConfirm\}$

In particular we can evaluate at any time how many user components have already completed the passage by: $Done(t) = RanUser(t) + RanBrowsing(t) + RanBuying(t)$ and therefore the CDF at time t is: $\frac{Done(t)}{RunningBrowsing(0)}$

However this would calculate a pessimistic CDF due to the second problem identified above. The solution is to use the results of steady-state analysis on the original model to 'seed' the probed model. We performed steady-state analysis on the original model and obtained the values for the populations of the user states shown in the table on the left of Figure 4.

The table on the right of Figure 4 shows how we 'seed' the probed model in order to achieve our cumulative distribution function. All those user states which are not the source state have their populations given to the state which corresponds to a probe which has already finished, in this case the steady-state population of the $User$ state is given to $RanUser$ and similarly for $Buying$. As mentioned before the $ProbeUser$ and $ProbeBuying$ states are never reached

User	300
Browsing	54
Buying	646

RunningUser	0
RunningBrowsing	54
RunningBuying	0
RanUser	300
RanBrowsing	0
RanBuying	646

Figure 4: The table on the left shows the steady-state populations of the user component states. The table on the right shows how the derived ODEs are seeded in order to obtain a CDF for the response-passage.

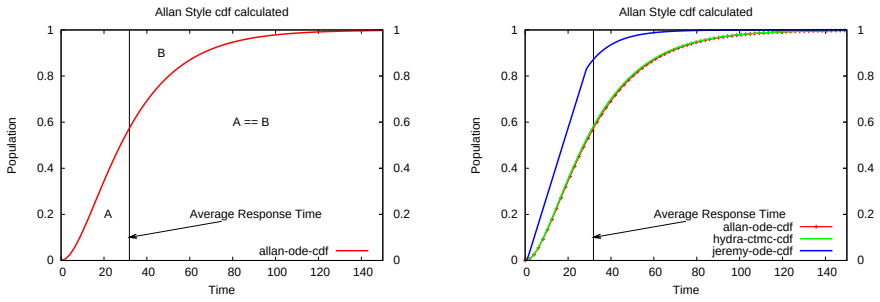


Figure 5:

and hence are given zero population. The source state *Browsing* has all of its steady-state population given to the equivalent *Browsing* state in which the probe is in the *Running* state. We can now compute the CDF of our response-passage by calculating the number of components initially in the source state which have now finished. This is: $Done(t) = (RanUser(0) + RanBuying(0)) - (RanUser(t) + RanBrowsing(t) + RanBuying(t))$ and we can compute the CDF by: $\frac{Done(t)}{RunBrowsing(0)}$

Using this we obtained our new CDF depicted in the left hand graph of Figure 5. The right hand graph compares the CDFs computed by our proposed method 'allan-ode-cdf' and the previous method of Bradley et al 'jeremy-ode-cdf' and that computed using a CTMC and a model used to approximate the true model which space does not permit to show here.

3 Conclusions

This paper has briefly shown how to update our ability to calculate a response-time profile from a PEPA model whose state space is too large to consider compilation via a CTMC. In addition we have shown how both absorbing probes and one-run probes may be used to alter the behaviour of a model, this is something of a first for the stochastic probe framework which has hitherto maintained a strict protocol that a stochastic probe must not alter the behaviour of the model.

References

[1] Hillston, J.: A Compositional Approach to Performance Modelling. Cambridge University Press (1996)

[2] Ribaudo, M.: On the aggregation techniques in stochastic Petri nets and stochastic process algebras. In Gilmore, S., Hillston, J., eds.: Proceedings of the Third International Workshop on Process Algebras and Performance Modelling, Special Issue of *The Computer Journal*, 38(7) (1995) 600–611

[3] Gilmore, S., Hillston, J., Ribaudo, M.: An efficient algorithm for aggregating PEPA models. IEEE Transactions on Software Engineering **27**(5) (2001) 449–464

[4] Hillston, J.: Fluid flow approximation of PEPA models. In: Proceedings of the Second International Conference on the Quantitative Evaluation of Systems, Torino, Italy, IEEE Computer Society Press (2005) 33–43

[5] Gillespie, D.: Exact stochastic simulation of coupled chemical reactions. Journal of Physical Chemistry **81**(25) (1977) 2340–2361

[6] Bradley, J., Gilmore, S.: Stochastic simulation methods applied to a secure electronic voting model. Electr. Notes Theor. Comput. Sci. **151**(3) (2006) 5–25

[7] Grassmann, W.: Transient solutions in Markovian queueing systems. Computers and Operations Research **4** (1977) 47–53

[8] Gross, D., Miller, D.: The randomization technique as a modelling tool and solution procedure for transient Markov processes. Operations Research **32** (1984) 343–361

[9] Clark, A., Gilmore, S.: Terminating passage-time calculations on uniformised Markov chains. In Argent-Katwala, A., Dingle, N.J., Harder, U., eds.: Proceedings of the Twenty-Fourth annual UK Performance Engineering Workshop. (2008) 64–75

[10] Little, J.D.C.: A proof of the queueing formula $l = \lambda w$. Operations Research **9** (1961) 380–387

[11] Clark, A., Duguid, A., Gilmore, S., Tribastone, M.: Partial evaluation of PEPA models for fluid-flow analysis. In Thomas, N., Juiz, C., eds.: Proceedings of the 5th European Performance Engineering Workshop (EPEW 2008). Volume 5261 of LNCS., Palma de Mallorca, Spain, Springer (2008) 2–16

[12] Bradley, J.T., Hayden, R., Knottenbelt, W.J., Suto, T.: Extracting Fluid Response times from PEPA models. In: PASTA'08, 7th Workshop on Process Algebra and Stochastically Timed Activities. (2008) This is a short version summary of the full paper that appeared at SIPEW 2008 (http://pubs.doc.ic.ac.uk/responsetimes-fluidanalysis).

[13] Clark, A., Gilmore, S.: State-aware performance analysis with eXtended Stochastic Probes. In Thomas, N., Juiz, C., eds.: Proceedings of the 5th European Performance Engineering Workshop (EPEW 2008). Volume 5261 of LNCS., Palma de Mallorca, Spain, Springer (2008) 125–140

Configuring Service-Oriented Systems using PEPA and AI Planning

Amanda Coles, Andrew Coles
Department of Computer and Information Sciences,
University of Strathclyde, Glasgow, UK
firstname.lastname@cis.strath.ac.uk

Stephen Gilmore
School of Informatics,
University of Edinburgh, UK
firstname.lastname@ed.ac.uk

August 14, 2009

Abstract

In this paper, we look at how two hitherto-distinct approaches to system modelling — PEPA and AI Planning — can be combined to provide configuration decisions for service-oriented systems. By combining the modelling of uncertainty in PEPA with the fast decision making of planning, we exploit their respective strengths. To illustrate the potential of our approach, we consider a model of a system for obtaining loan decisions from a broker, for which we produce cost-effective investment decisions to meet a desired performance target.

1 Introduction

The cost-effective provision of Service-Oriented Systems — 'systems of systems' set up to provide a desired service — presents an interesting challenge to 21st century computing. Considering two facets of the problem, first, there is the challenge of modelling expected system performance given the performance of system components and network infrastructure. Second, there is the issue of choosing the system configuration: the possibility of distributing data processing on a global scale gives considerable choice over how to compose the service, and as such finding the most cost effective system configuration to meet a desired performance target is non-trivial. For instance, a cheaper remote service is only cost-effective if the investment in bandwidth needed to meet the desired performance is reasonable; but at the same time, investing elsewhere in the system may make investment in bandwidth unnecessary.

In this paper, we explore how two approaches to system modelling can be combined to meet these challenges. For an accurate performance model, we will use PEPA [4]. For fast decision making, we will use AI Planning [2], working with an approximate model of system performance. The two systems, combined, form a closed loop: when the model of performance provided initial PEPA model is unacceptable, planning is used to provide cost-effective investment decisions; then, an updated PEPA model is built to ascertain whether the performance target has now truly been met.

2 Background: AI Planning

AI Planning is concerned with the task of finding a sequence of actions that, when executed, reach some desired goals. Beginning with planning in its simplest form — propositional planning, without an an explicit notion of time or numbers — a planning problem can be described by a tuple $\langle I, G, A \rangle$, where:

- I is the *initial state*: a set of propositions that hold true initially (under the closed world assumption that any fact not in I is initially false);
- G is the *goal state*: a set of propositions such that $G \subseteq S_g$ for any goal state S_g;
- A is the set of *actions*, with the aim being to reach a state S_g through applying successive actions from A to I.

Each action $a \in A$ can, in turn, be defined by a tuple $\langle pre, del, add \rangle$, where:

- pre is the *precondition* set of a: for a to be applied in a state S, $pre \subseteq S$;
- del and add are the set of facts deleted (resp. added) upon the application of a.

```
(:action drive-truck
 :parameters (?t - truck ?from ?to - locations)
 :precondition (and (at ?t ?from)
                 (>= (fuel ?t) (/ (distance ?from ?to) (mpg ?t)))
               )
 :effect
   (and (not (at ?t ?from))
        (at ?t ?to)
        (decrease (fuel-in ?t) (/ (distance ?from ?to) (mpg ?t)))
   )
)
```

Figure 1: Moving a truck

Applying a in a state S then gives a state S' where:

$$S' = (S \setminus del(A)) \cup add(A)$$

More recently [3], planning models written in the Planning Domain Definition Language (PDDL) can include numeric state values in their specification. An example of an action manipulating numeric values is shown in Figure 1. As can be seen, for the action to be applied, the truck must be at the designated start location, and hold enough fuel for the journey; and when the action is applied, the location of the truck is updated, and the amount of fuel in it decreased accordingly. With models such as these, the task of a modern planner such as COLIN [2] is to find a solution plan that respects this defined behaviour, along with other capabilities of PDDL, such as modelling time and continuous numeric processes (e.g. filling a tank at a certain rate).

3 Modelling Service-Oriented Systems in PEPA

We will use PEPA [4] to model a service-oriented system where a customer (CR) requests loans from a broker (BR), who forwards on the request to a lender (LE), and returns the response to the customer. The customer, broker and lender are physically distributed and communicate across a network between the customer and the broker (CB) and a network between the broker and the lender (BL).

The Service-Level Agreement (SLA) for this system is expressed in terms of the response time experienced by the customer. That is, the delay between the end of the request activity and the end of the response activity. (It does not matter how long the customer takes to make a request, we are concerned only with how responsive is the service composition of broker and lender, once the request has come in.) The SLA of the loan service states that 80% of customers receive a response within 7 seconds. The SLA does not state what percentage of these responses turn down the loan request.

The customer (CE) makes a request and then is ready to receive a loan offer. When this is completed, the customer thinks about the offer, and the process repeats.

$$\begin{aligned} CR &\stackrel{def}{=} (\mathit{request}, r).CR_{ready} \\ CR_{ready} &\stackrel{def}{=} (\mathit{response}, \top).CR_{completed} \\ CR_{completed} &\stackrel{def}{=} (\mathit{think}, t).CR \end{aligned}$$

The network between the customer and the broker is symmetric in the sense that it is reasonable to use only a single rate parameter r_{CB} to represent transfer from customer to broker in both directions (first for the request, then for the response).

$$\begin{aligned} CB \stackrel{def}{=}\ & (\mathit{request}, \top).(\mathit{transferToBroker}, r_{CB}). \\ & (\mathit{transferFromBroker}, \top).(\mathit{response}, r_{CB}).CB \end{aligned}$$

The broker (BR) is idle until a loan request comes in. The broker then identifies a suitable lender by consulting a registry and then forwards the loan request to the lender. (The interaction with the registry is not represented here.) When the response is received from the lender the broker post-processes it and transfers it back across the network to the customer.

$$\begin{aligned} BR \stackrel{def}{=}\ & (\mathit{transferToBroker}, \top).(\mathit{requestLender}, r_{BR}). \\ & (\mathit{responseLender}, \top).(\mathit{transferFromBroker}, r_{BR}).BR \end{aligned}$$

The network between the broker and the lender is also symmetric, imposing the same delay when transferring the request to the lender, as when transferring the response from the lender. Thus the rate r_{BL} is used twice in the definition of this

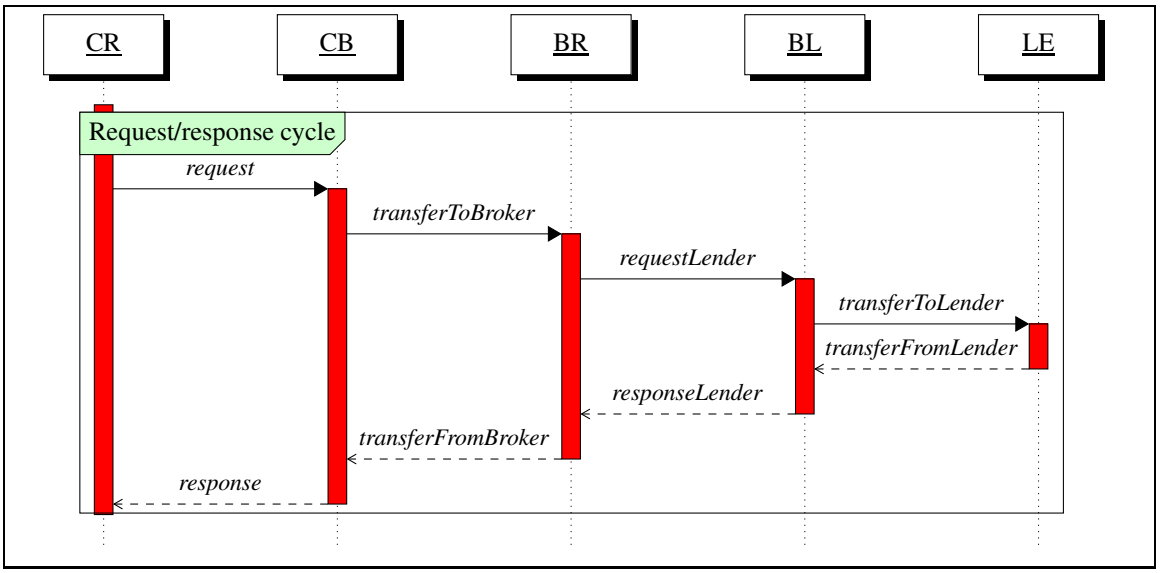


Figure 2: A UML sequence diagram showing the interactions between the customer (CB), the broker (BL) and the lender (LE).

component.

$$BL \stackrel{def}{=} (\mathit{requestLender}, \top).(\mathit{transferToLender}, r_{BL}).\\ (\mathit{transferFromLender}, \top).(\mathit{responseLender}, r_{BL}).BL$$

The lender (LE) simply functions as a reactive system, being inactive until receiving a request and then giving back the loan offer.

$$LE \stackrel{def}{=} (\mathit{transferToLender}, \top).(\mathit{transferFromLender}, r_{LE}).LE$$

Finally, the complete PEPA model is a composition of these five sequential components, requiring them to cooperate on their common actions.

$$(CR \underset{\mathcal{L}_1}{\bowtie} (CB \underset{\mathcal{L}_2}{\bowtie} (BR \underset{\mathcal{L}_3}{\bowtie} (BL \underset{\mathcal{L}_4}{\bowtie} LE))))$$

where

$$\begin{aligned} \mathcal{L}_1 &= \{\mathit{request}, \mathit{response}\} \\ \mathcal{L}_2 &= \{\mathit{transferToBroker}, \mathit{transferFromBroker}\} \\ \mathcal{L}_3 &= \{\mathit{requestLender}, \mathit{responseLender}\} \\ \mathcal{L}_4 &= \{\mathit{transferToLender}, \mathit{transferFromLender}\} \end{aligned}$$

The activity relevant to the SLA is depicted as a UML diagram in Figure 2. The five sequential components of the PEPA model are represented as swimlanes which proceed down the page. The horizontal lines which cross from one swimlane to another mark the end of an activity. The height of a thick bar represents the duration of one or more activities. We are concerned with the height of the bar in the CB swimlane, which represents the seven-activity sequence *transferToBroker*; *requestLender*; *transferToLender*; *transferFromLender*; *responseLender*; *transferFromBroker*; and *response*. These activities need to take place (in this order) to complete a request from a customer.

4 Approaching PEPA Models from Planning

As discussed in the previous section, PEPA provides an elegant means whereby service-oriented systems can be formally defined. Its key strength in this setting lies in the ability to use distributions to model the how long each activity takes, capturing the uncertainty in the performance of each of the system's components. In doing so, it is possible to verify service-level agreements (SLA) specifying a total service time and the percentage of transactions for this must hold. The weakness of the approach is encountered when the service-level agreement is not met, and hence where investment is needed to reduce one or more activities' durations. There is no direct way to consider these possibilities: the modeller is left with the task of performing sensitivity analyses on the system to deduce where investment should be applied, whilst bearing in mind the costs of doing so.

The strengths and weaknesses of planning models contrast strongly with that of PEPA in this domain. As discussed in Section 2, a PDDL model can capture a sequence of activity, but unlike PEPA, in classical planning the outcome of the

```
(:action CB1
 :parameters ()
 :precondition
      (ready)
 :effect (and
      (not (ready))
      (ready_for br1)
      (increase (dur) 0.3)
   )
)
```

```
(:action BR1
 :parameters ()
 :precondition
      (ready_for br1)
 :effect (and
      (not (ready_for br1)))
      (ready_for bl1)
      (increase (dur) 0.9)
   )
)
```

Figure 3: CB1 and BR1 Actions from the Basic PDDL model

action is deterministic: moving the truck uses precisely the calculated amount of fuel, and were a temporal model used, it would take precisely the calculated amount of time. Where planning has its strengths is in considering the implications of decisions and how well they contribute towards the goal. Actions can be used to encode the options available and their costs, and the planner can then consider these when planning to reach the goals. For instance, in the case of a logistics problem, the costs would correspond to refueling the trucks and employing drivers.

From these comparisons, what we seek is useful middle ground combining the strengths of these two approaches. Where a PEPA model falls short of the SLA, investment decisions are needed — a strength of planning. But, to make useful investment decisions, the planner needs to consider uncertainty — a strength of PEPA models. Here, we shall proceed to answer the following: how can we give a planner enough information about the uncertainty in activities' durations to allow it to make cost-effective service investment decisions, leading to a modified PEPA model which is then feasible?

4.1 A Basic PDDL System Model

First, we shall construct a basic PDDL model corresponding to the steps of the GetLoan system (Figure 2). Our initial state declares that the system is idle (the fact `ready` holds), and the goals are that the second CB activity has finished, and the total time taken is ≤ 6 seconds. For each activity, a corresponding action is created in the planning problem; for activities in two fragments (e.g. CB) we add two actions, $CB1$ and $CB2$. Two example actions are shown in Figure 3. First, considering numeric effects, each action increases the duration of the activity sequence by the mean time taken to complete the activity. Second, considering the propositional preconditions and effects, the first of these adds a fact required as a precondition for the second. Similar effect–precondition constructs exist for the remaining actions, until $CB2$ adds `done`, as required by the goal state.

4.2 Modelling Uncertainty and Choice

The PDDL model, as described, only considers the mean time taken to complete each activity, taking the total time to complete the sequence as the sum of these. As there is no model of uncertainty, the planner has no basis on which to determine the distribution of possible outcome durations, and hence no means to determine whether the SLA will hold under the exponential distributions used as the basis for the original PEPA model. To address this, we shall approximate these distributions within the PDDL model, allowing the planner to estimate the attainable SLA.

To achieve this, we make two changes to our model. First, each action has a number of duration outcomes, associated with likelihoods, rather than a single duration effect. For durations samples $d_0, .., d_n$ taken from the time distribution for activity a, the corresponding likelihood $p_i \in p_0, .., p_n$ will be taken as $(CDF(d_i, \lambda_a) - CDF(d_{i-1}, \lambda_a))$ (defining $CDF(d_{-1}, \lambda_a) = 0$). Second, each action applies its outcomes onto each of the previous possible outcomes. In doing so, once the plan contains all the necessary activities, by summing the likelihood of all outcomes within the target deadline (e.g. 6 seconds), we can see if the target likelihood (e.g. 90%) has been met.

Before illustrating this with an example, we make one final change to the model to reflect choice over the investment decisions available to improve the system's performance. For each activity, a range of rates are available, with associated costs. By planning to minimising the sum of these costs, whilst also meeting the SLA, the planner can answer questions of the form 'what is the minimum investment needed to meet this SLA?'. Figure 4 shows an example of the finished action model. The cost effect is fairly intuitive — increase the total cost by the cost of setting $BR1$ to follow the rate chosen. Also note the effect `(ratefor_br2 ?r)`, which records that this rate must later be used for $BR2$.

The outcome effects are slightly more involved. At a superficial level, whenever an action is applied, we update the space of possible outcome durations and likelihoods for the plan so far (as a whole) to reflect the space of possible outcomes from the action. In more detail, the rationale behind the PDDL nested `forall` effects are as follows:

```
(:action BR1
 :parameters (?r - rate)
 :precondition
      (ready_for br1)
 :effect (and
      (not (ready_for br1))
      (ready_for bl1)
      (increase (cost) (cost_br1 ?r))
      (ratefor_br2 ?r)
      (forall (?sa - sample_cb1)
        (forall (?sb - sample_br1) (and
           (assign (outcome_d_br1 ?sa ?sb) (+ (sample_d_br1 ?r ?sb) (outcome_d_cb1 ?sa)))
           (assign (outcome_p_br1 ?sa ?sb) (* (sample_p_br1 ?r ?sb) (outcome_p_cb1 ?sa)))
        ))
      )
   )
)
```

Figure 4: CB1 from the Enhanced PDDL model

1. The earlier action for $CB1$ defined a number of possible outcomes, the set `sample_cb1`. The duration of each of these (i.e. each `?sa - sample_cb1`) is denoted `(outcome_d_cb1 ?sa)`, and its associated likelihood is `(outcome_p_cb1 ?sa)`.

2. The action $BR1$ also has a number of possible outcomes, the set `sample_br1`. The rate parameter defines the space of outcome durations and likelihoods, so for each `?sb - sample_br1` there is an outcome `(sample_d_br1 ?sb)` with likelihood `(sample_p_br1 ?sb)`.

3. The number of possible outcomes from $CB1$ then $BR1$ is then the cartesian product of the outcome sets, summing the durations and multiplying the likelihoods.

5 Test Case

As a proof-of-concept for the use of planning to make cost-effective investment decisions to meet a desired SLA, we shall refer to the GetLoan system (Figure 2). The scenario is as follows:

- The existing system is (easily) capable of meeting an SLA of 7 seconds 80% of the time.
- The goal is to meet an SLA of 6 seconds 90% of the time, requiring investment.
- The current mean delays for each of CB,BR,BL and LE are 0.3, 0.9, 0.3 and 1.2 seconds, respectively;
- The delay for each of these can be reduced by 20%, with respective costs of £10k, £15k, £8k, and £18k.

We proceed to make a PDDL model of this, following Section 4.2. We take 10 samples from each activity's time distribution using the baseline delays, adding these to the model with cost 0. Then, reducing the delays by 20%, we take a further set of samples and add these as alternatives, incurring the relevant cost. The plan produced is as follows, with a total investment cost of £23k:

1. CB1, baseline rate
2. BR1, improved rate
3. BL1, improved rate
4. LE, baseline rate
5. BL2, improved rate
6. BR2, improved rate
7. CB2, baseline rate

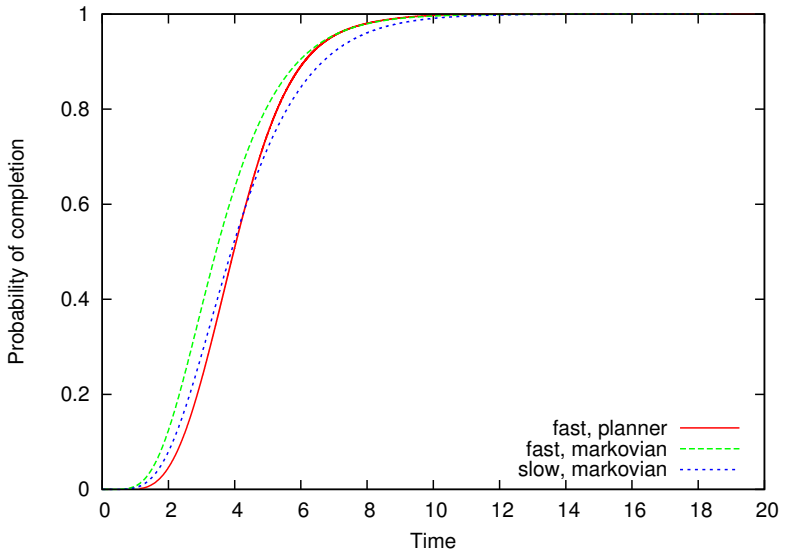


Figure 5: CDFs of the Performance of the GetLoan System

The time taken to find the plan was two seconds, and is known to be cost-optimal, with respect to the planning model: if we add an additional goal that cost is less than 23k, no solution plan can be found.

Bearing in mind that the outcome distribution used by the planner is an approximation, we can then update the original PEPA model to validate whether the investment decisions made by the planner are able to meet the new SLA. We performed a Markovian response-time analysis of the PEPA model using the ipclib [1] PEPA library, and the results for the original and updated models are shown in Figure 5 (the 'slow, markovian' and 'fast, markovian lines', respectively). The 6 second lies at the 90.544%th percentile of the updated CDF, indicating the investment has reached the desired SLA. Also shown on the graph is the estimated CDF produced by the planner for the outcomes of the above plan — as can be seen, it is pessimistic, but within a few percentage points of the true CDF in the upper quartile.

6 Conclusions

In this paper we have shown how PEPA modelling and Planning can be usefully combined to provide decision-support for service-oriented system configuration. By using a PEPA model to accurately reflect the uncertainty in the time taken to complete each system activity, and a planner working with an approximate sampled model, the result is a CPU-time-efficient system for suggesting and validating system investment decisions. In future work, our aim is to extend the range of system models for which our approach can be used by extending the planner to support more flexible PEPA models with disjunctive activity pathways, as well as larger models with greater numbers of possible activity configurations.

Acknowledgements

Amanda Coles is supported by the EPSRC project EP/G023360/1, "Modelling Planning Problems". Andrew Coles is supported by SICSA, the Scottish Informatics and Computer Science Alliance. Stephen Gilmore is supported by the EU FET-IST Global Computing 2 project SENSORIA ("Software Engineering for Service-Oriented Overlay Computers" (IST-3-016004-IP-09)).

References

[1] Allan Clark. The ipclib PEPA Library. In Mor Harchol-Balter, Marta Kwiatkowska, and Miklos Telek, editors, *Proceedings of the 4th International Conference on the Quantitative Evaluation of SysTems (QEST)*, pages 55–56. IEEE, September 2007.

[2] A. J. Coles, A. I. Coles, M. Fox, and D. Long. Temporal planning in domains with linear processes. In *Twenty-First International Joint Conference on Artificial Intelligence (IJCAI)*, July 2009.

[3] M. Fox and D. Long. PDDL2.1: An Extension of PDDL for Expressing Temporal Planning Domains. *J. Art. Int. Research*, 20:61–124, 2003.

[4] J. Hillston. *A Compositional Approach to Performance Modelling*. Cambridge University Press, 1996.

Abstraction and Model Checking in the Eclipse PEPA Plug-In

Michael J. A. Smith*
M.J.A.Smith@sms.ed.ac.uk
Laboratory for Foundations of Computer Science
University of Edinburgh
Edinburgh, United Kingdom

Abstract

The stochastic process algebra PEPA is a widely used language for performance modelling, and a large part of its success is due to the rich tool support that is available. As a compositional Markovian formalism, however, it suffers from the state space explosion problem, where even small models can lead to very large Markov chains. One way of analysing such models is to use abstraction — constructing a smaller model that bounds the properties of the original.

We present a new tool for abstracting and model checking PEPA models, which is an extension of the Eclipse plug-in for PEPA. This extends the current tool with two additional views. The abstraction view is a graphical interface for labelling and aggregating states of PEPA components. The model checking view allows CSL properties to be specified and checked. Internally, we use the techniques of abstract Markov chains and stochastic bounds to analyse transient and steady state properties respectively. We have extended both techniques so that they can be applied compositionally to PEPA.

1 Introduction

In the distributed world of today, performance has a vital role in the construction of robust, reliable and scalable computer systems. A powerful technique for reasoning about the performance of these systems is to use mathematical modelling — in particular, working with high-level compositional languages, such as stochastic process algebras. One such widely used language is the Performance Evaluation Process Algebra (PEPA) [10], and a significant part of its success is due to the many tools that are available [5, 9, 15, 20].

The PEPA language has two primary semantics — a Markovian semantics, which maps a model to a continuous time Markov chain (CTMC) [10], and an ordinary differential equation (ODE) semantics [11]. Both semantics, as well as stochastic simulation, are implemented in the PEPA plug-in tool [20], which is built on the Eclipse platform [1]. This provides an interface for editing PEPA models, and performing both steady state and time series analysis. An Integrated Development Environment (IDE) such as Eclipse has the advantage of providing a standardised interface, with which users are already familiar, and removing much of the burden of user interface development from the tool developer.

Often, when we analyse a model, we are interested in a particular property that we express in an appropriate logic — for example, the Continuous Stochastic Logic (CSL) [3, 4]. There are a number of tools available for model checking Markovian models — for example, MRMC [13] and PRISM [15] — but currently PRISM is the only model checker that accepts PEPA as a direct input language. Whilst PRISM is a powerful tool, it has two disadvantages for the PEPA modeller — it accepts only a subset of the PEPA language[1], and it is not integrated with the PEPA plug-in tool. One of the contributions of this paper is to present *integrated* tool support for CSL model checking of the *full* PEPA language.

The main problem faced by compositional formalisms like PEPA, is that even a relatively small model can lead to a CTMC that is much too large to analyse. This state space explosion problem can be avoided if we are only interested in the average behaviour of the system — namely, the average number of components in a given state at a particular time — in which case we can use the ODE semantics of PEPA. However, if

*This work was funded by a Microsoft Research European Scholarship

[1]PRISM does not implement the minimum semantics of active-active synchronisation, and instead multiplies the rates.

we want to reason about *all* possible runs of the model, and not just the average case, we need to look at its Markovian semantics. And since the problem of state space explosion is unavoidable, we have no choice but to look for ways of dealing with it.

The basic principle for analysing a large model seems deceptively simple — we just need to make it smaller! This process of *abstraction* is more difficult than it sounds, because we need to preserve information about the properties we are interested in. There a few different ways for us to do this, but the approach we take in this paper is to *aggregate* states with similar behaviour. Since, most of the time, this throws away some of the information in the original model, we look at obtaining *bounds* on properties of the model. For example, if a system has a probability 0.001 that it fails within the first ten minutes of operation, the abstracted model might tell us that the probability lies in the interval $[0, 0.002]$. The important thing is that the bound is *accurate*, in that the actual probability always lies within the interval we obtain.

Whilst some abstractions can give very tight bounds on the probability of satisfying a property, others can give less useful information. Indeed, in the worst case, we could "discover" that the probability lies in the interval $[0, 1]$. Hence the key to obtaining useful information is the choice of abstraction — in other words, how do we select which states to combine? As there is currently no way, in general, to automatically decide this, it is important that we can easily try different abstractions. This leads to the topic of this paper — providing an interface for quickly and easily specifying abstractions of PEPA models, and for checking properties of the abstracted models.

There are a number of techniques for producing bounded abstractions of a CTMC, and we make use of two of them — *abstract Markov chains* [7,13], and *stochastic bounds* [8,19]. The former can be used to bound transient properties, and the latter for various monotone properties such as the steady state distribution. Both of these techniques were originally specified and used at the level of Markov chains, but we have extended them so that they can be applied *compositionally* to PEPA models [18]. This means that we can construct bounds for models whose underlying CTMC is too large to represent (let alone analyse), since we maintain its compositional structure when building the abstract model. It also means that the modeller can identify which states to aggregate in a more natural way, using the same component-level view as in the original model.

In this paper, we present a new extension to the Eclipse PEPA plug-in, which provides a graphical interface for abstracting PEPA models, and a model checker for properties in the Continuous Stochastic Logic (CSL) [3,4]. The tool can be downloaded from `http://www.dcs.ed.ac.uk/pepa/tools/plugin`, and extends the PEPA plug-in with the following two views:

1. The **Abstraction View** is a graphical interface that shows the state space of each sequential component in a PEPA model. It provides a facility for labelling states (so that they can be referred to in CSL properties), and for specifying which states to aggregate.

2. The **Model Checking View** is an interface for constructing, editing, and model checking CSL properties. The property editor provides a simple way to construct CSL formulae, by referencing states that are labelled in the abstraction view. Only valid CSL formulae can be entered.

In this paper, we will begin by describing the PEPA language, along with a small example, in Section 2. We will then describe the abstraction and model checking views in Sections 3 and 4 respectively. We give a brief description of the theory behind the approach, and some implementation details in Section 5. Finally, we consider a larger example in Section 6, to illustrate the capabilities of the tool, before concluding in Section 7.

2 The PEPA Language

The Performance Evaluation Process Algebra (PEPA) [10] is a compositional formalism with Markovian semantics. In PEPA, a *system* is a set of concurrent *components*, which are capable of performing *activities*. An activity $a \in \mathcal{A}ct$ is a pair (a, r), where $a \in \mathcal{A}$ is its action type, and $r \in \mathbb{R}_{\geq 0} \cup \{\top\}$ is the rate of the activity. This rate parameterises an exponential distribution, and if unspecified (denoted $\top$), the activity is said to be *passive*. In this case, another component must actively drive the rate of the action. PEPA terms

$$
\begin{array}{lcl}
P_{14} & = & (reg_{14}, r).P_{14} + (move_{15}, m).P_{15} \\
P_{15} & = & (reg_{15}, r).P_{15} + (move_{14}, m).P_{14} + (move_{16}, m).P_{16} \\
P_{16} & = & (reg_{16}, r).P_{16} + (move_{15}, m).P_{15}
\end{array}
$$

$$
\begin{array}{lcl}
S_{14} & = & (reg_{14}, \top).(rep_{14}, s).S_{14} \\
S_{15} & = & (reg_{15}, \top).(rep_{15}, s).S_{15} \\
S_{16} & = & (reg_{16}, \top).(rep_{16}, s).S_{16}
\end{array}
$$

$$
\begin{array}{lcl}
DB_{14} & = & (rep_{14}, \top).DB_{14} + (rep_{15}, \top).DB_{15} + (rep_{16}, \top).DB_{16} \\
DB_{15} & = & (rep_{14}, \top).DB_{14} + (rep_{15}, \top).DB_{15} + (rep_{16}, \top).DB_{16} \\
DB_{16} & = & (rep_{14}, \top).DB_{14} + (rep_{15}, \top).DB_{15} + (rep_{16}, \top).DB_{16}
\end{array}
$$

$$P_{14} \underset{\{reg_{14}, reg_{15}, reg_{16}\}}{\bowtie} (S_{14} \parallel S_{15} \parallel S_{16}) \underset{\{rep_{14}, rep_{15}, rep_{16}\}}{\bowtie} DB_{14}$$

Figure 1: A PEPA model of an active badge system

have the following syntax:

$$C := (a, r).C \mid C_1 + C_2 \mid C_1 \underset{L}{\bowtie} C_2 \mid C/L \mid A$$

We briefly describe these combinators as follows:

- *Prefix* ($(a, r).C$): the component can carry out an activity of type a at rate r to become component C.
- *Choice* ($C_1 + C_2$): the system may behave either as component C_1 or C_2. The current activities of both components are enabled, and the first to complete determines which component proceeds. The other component is discarded.
- *Cooperation* ($C_1 \underset{L}{\bowtie} C_2$): the components C_1 and C_2 synchronise over the cooperation set L. For activities whose action type is not in L, the two components proceed independently. Otherwise, they must perform the activity together, at the rate of the slowest component. At most one of the components may be passive with respect to this action type.
- *Hiding* (C/L): the component behaves as C, except that activities whose types are in L are hidden, and appear externally as the unknown type τ.
- *Constant* ($A \stackrel{def}{=} C$): component C has the name A.

The operational semantics of PEPA defines a labelled multi-transition system, which induces a *derivation graph* for a given component. Since the duration of a transition in this graph is given by an exponentially distributed random variable, this corresponds to a continuous time Markov chain (CTMC).

An example PEPA model with five components is shown in Figure 1. This is a model of an active badge sensor system, which was first presented in [6]. In the model, a person (component P) moves between three corridors, labelled 14, 15 and 16, which are arranged linearly. Each corridor i has a sensor S_i, which listens for a registration signal reg_i from the person, and informs the database DB. The state of the database effectively records where the person was last seen. The model has three rate parameters — r is the rate at which the badge sends a signal to the sensors, m is the rate of moving between corridors, and s is the rate at which the sensor updates the database.

There are a number of interesting questions we might ask of the model. For example, what proportion of the time does the person spend in each of the corridors? Or, what is the probability that the database can move directly from state DB_{14} to state DB_{16}, missing the fact that the person must have passed through corridor 15? Since this particular example has only 72 states, we can analyse it directly without need for abstraction. We will, however, use it as a running example whilst describing the tool in the following two sections. In Section 6 we will look at a larger example, which better illustrates the power of abstraction.

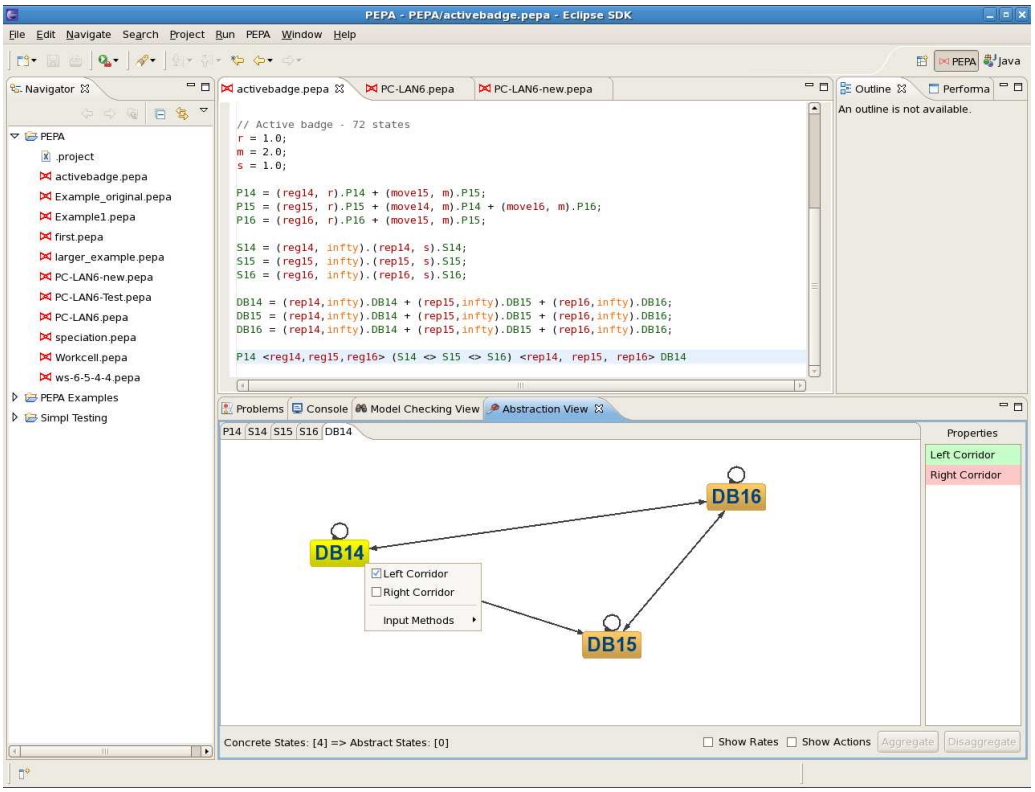


Figure 2: The PEPA Eclipse Plug-In

3 Specifying State-Based Abstractions

In order to construct and check CSL properties of a PEPA model, we need some way of referring to states of the model. One way of doing this would be use the names of the sequential component states in the model, but this could lead to very long and cumbersome names — especially if we refer to a large set of states. Ideally, we would prefer to use a single, meaningful name. Our solution is to provide a graphical interface for labelling sets of states.

An overall view of the PEPA plug-in is shown in Figure 2. Here, the active badge model of Figure 1 is open in the editor, and the abstraction view is in use. The majority of the abstraction view is taken up by a graphical representation of the sequential components in the system equation of the model. In this case, there are five components, and each corresponds to a tab in the view. Currently on display is the database component DB.

On the right of the abstraction view is a table showing the atomic properties for the model. Right clicking on this table brings up a menu, from which we can define a new property, or rename or delete an existing one. When we create a new property, the currently selected states in the graph will initially satisfy it, and the other states will not. We can change which properties a state satisfies by right clicking on the state — this allows us to select or deselect the atomic properties, as illustrated in the figure.

An additional feature of the property table is that clicking on a property will highlight all the states that satisfy it. Clicking on a state in the graph will shade the properties that it satisfies in green, and those it does not in red. This allows us to quickly see which states satisfy which properties. It is important to remember that all atomic properties are defined *compositionally*. A state in the system satisfies an atomic property if and only if its state in each component does. In Figure 2, the property "Left Corridor" is satisfied by DB_{14}, but not by DB_{15} and DB_{16}. Since we do not constrain the property for any of the other components, it is satisfied by all states of the system that have the database in state DB_{14}. An example would be the state $P_{14} \parallel S_{14} \parallel S_{15} \parallel S_{16} \parallel DB_{14}$.

The second function of the abstraction view is, as its name suggests, to specify an *abstraction* — namely, which states to aggregate. Figure 3 shows a close-up of the abstraction view, this time for the component P.

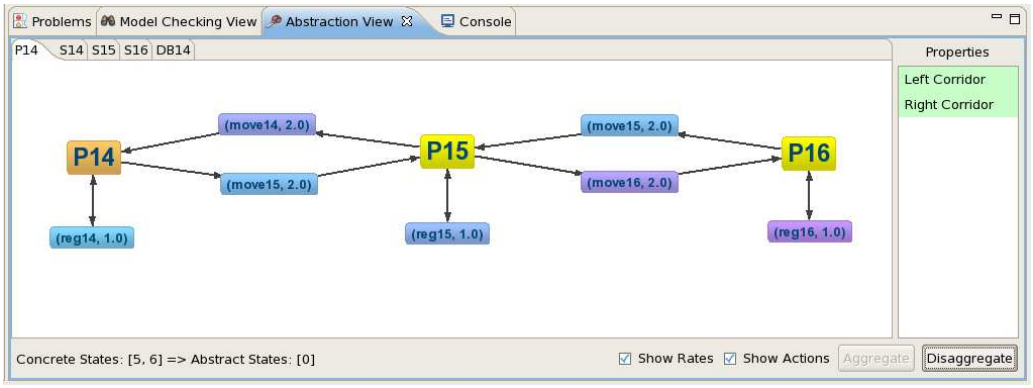


Figure 3: The abstraction interface

In this case, we have selected to show both the actions and the rates on the transitions, but since this leads to a more cluttered graph these options are not selected by default.

Aggregating states in a sequential component is simply a matter of selecting the states, and clicking the *Aggregate* button. They can be separated again by clicking *Disaggregate*. Once a set of states have been aggregated, we can only select them as a group — clicking on any one of the states will select them all. Note that the aggregation of states is independent of both the labelling of atomic properties[2] and the definition of CSL properties in the model checking view. This means that we can quickly try out different abstractions — the only thing we need to do is to run the model checker each time.

4 Model Checking Abstract PEPA Models

To describe properties of a PEPA model, we need a logic for expressing them. The most widely used logic for model checking CTMCs is the Continuous Stochastic Logic (CSL) [3, 4]. CSL is a branching-time temporal logic, which allows us to talk about the *probability* of a state satisfying some temporal property, and the *time interval* in which a property must hold.

Formulae in CSL consist of *state formulae* Φ, and *path formulae* φ. The former are properties of individual states in the Markov chain — for example, that the steady state probability is greater than a certain value. The latter are properties that hold of paths (sequences of states) through the chain — for example, that a state property holds until some condition is met.

State formulae Φ are defined as follows, for $\trianglelefteq \in \{\leq, \geq\}$, $p \in [0, 1]$ and atomic propositions a:

$$\Phi ::= \mathtt{tt} \mid a \mid \Phi \wedge \Phi \mid \neg\Phi \mid \mathcal{S}_{\trianglelefteq p}(\Phi) \mid \mathcal{P}_{\trianglelefteq p}(\varphi)$$

In addition to atomic propositions and the standard logical connectives, there are two state formulae of interest:

- A *steady state measure* $\mathcal{S}_{\trianglelefteq p}(\Phi)$ is satisfied if the steady state probability of being in the set of states satisfying Φ is $\trianglelefteq p$.
- A *path measure* $\mathcal{P}_{\trianglelefteq p}(\varphi)$ is satisfied of a state s if the integral of the probability measures of all the paths from s satisfying φ is $\trianglelefteq p$.

A useful extension to the basic CSL syntax, used by the PRISM [15] model checker and by us, allows us to test the value of a steady state or path measure:

$$\Phi_T = \mathcal{S}_{=?}(\Phi) \mid \mathcal{P}_{=?}(\varphi)$$

[2] If we aggregate a set of states when only some of them satisfy a property, the abstract state will have a truth value of '?' (i.e. it 'maybe' satisfies the property).

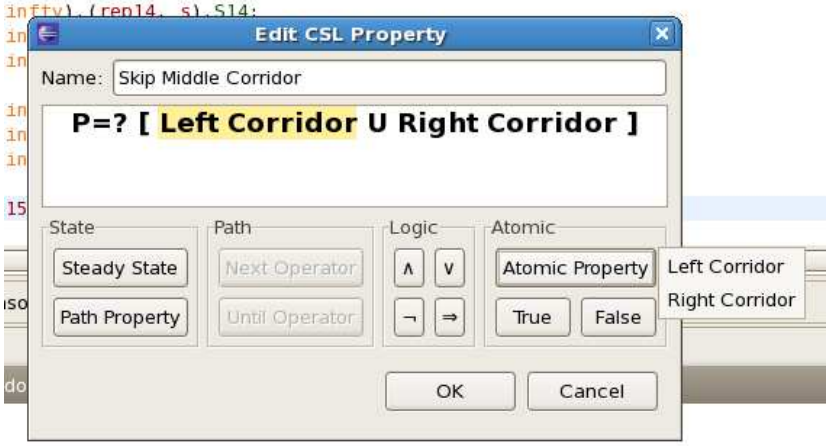


Figure 4: The CSL property editor

These are not state formulae in themselves, since they do not evaluate to a truth value, but to the *probability* of the property holding of a state. This does not affect the expressivity of CSL, but is convenient for users of a model checker.

Path formulae φ have the following syntax, where I is a non-empty interval over $\mathbb{R}_{\geq 0}$:

$$\varphi ::= X^I \Phi \mid \Phi \; \mathcal{U}^I \Phi$$

These two operators have the following semantics:

- The *next* operator, $X^I \Phi$, holds of a path if it leads to a state satisfying Φ in one transition, within the time interval I.
- The *until* operator, $\Phi_1 \; \mathcal{U}^I \Phi_2$, holds of a path if it reaches a state satisfying Φ_2 within the time interval I, and until then only passes through states that satisfy Φ_1.

Classically, for a CTMC, a CSL formula will evaluate to either true or false, or a probability in the case of the test operators Φ_T. In our case, however, we want to model check *abstract* models, hence the test operators will evaluate to a *probability interval*. This describes the best and worst case probability of satisfying the formula, based on the information available in the abstraction. Since this means that we might not know whether or not the model satisfies a property, we need to use a three-valued semantics of CSL [13], with truth values of $\top$, $\bot$ and ? (true, false, and maybe).

To illustrate the properties we can express, consider the following formula, for the active badge model from Figure 1. Let us assume that we used the abstraction view to define the atomic properties *Left Corridor* and *Right Corridor*, meaning that the database is in states DB_{14} and DB_{16} respectively:

$$\mathcal{P}_{=?}(\mathit{Left\ Corridor}\ \mathcal{U}\ \mathit{Right\ Corridor})$$

This asks the question, "what is the probability that the database will continue to think that the person is in the leftmost corridor, until it becomes aware that the person is in the rightmost corridor?" We can construct this formula using the CSL editor, as illustrated in Figure 4.

The aim of the CSL editor is to make it as easy as possible to construct a CSL formula. In particular, it ensures that we can construct only valid formulae. The buttons on the interface correspond to the various CSL operators and logic connectives, and are enabled by clicking on the part of the formula we want to edit. Hence we cannot enter a path formula where a state formula is required, or vice versa, and the test operators Φ_T can only be used at the top level of a formula. The path and state operator buttons produce a pop-up menu with the available choices — for example, timed versus untimed until operators.

The most useful feature of the CSL editor is that it presents us with a list of the atomic formulae that we defined in the abstraction view. Hence, we can easily refer to sets of states in the model, using the labels

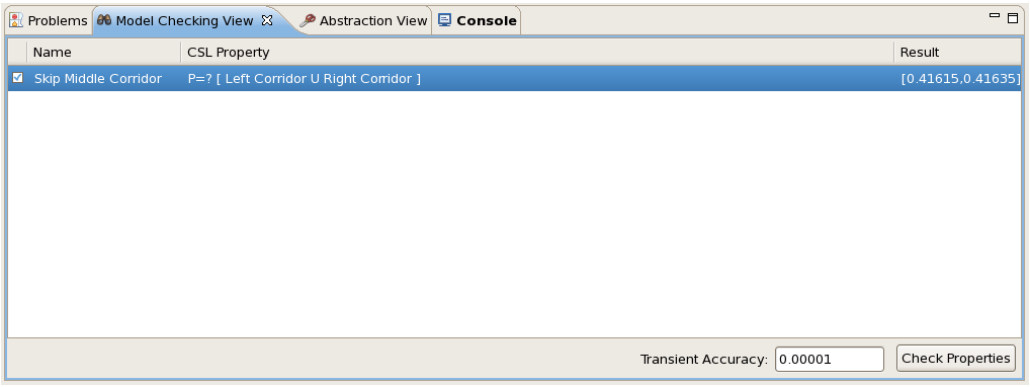


Figure 5: The model checking interface

```
13:16:04 [badge.pepa] Model added.
13:16:04 [badge.pepa] Model parsed.
13:16:08 [badge.pepa] Kronecker state space derived.  Elapsed time:  5 ms.
13:24:36 [badge.pepa] <Model Checker> Generating abstract CTMC...
13:24:36 [badge.pepa] <Model Checker> Optimising uniformisation constant to 7.0...
13:24:36 [badge.pepa] <Model Checker> Generated abstract CTMC with 72 states.
13:24:36 [badge.pepa] Property "P=?  [ Left Corridor U Right Corridor ]" was checked in 28 ms.
```

Figure 6: Console output from the model checker

we created. Because the internal data structures are shared, if we change the name of a property in the abstraction view, it will automatically be updated in the CSL formulae that use it. Similarly, a property cannot be deleted from the abstraction view while it is being used in a formula.

Figure 5 shows the model checking view, from which the CSL editor can be opened. The main component of the view is a table of all the properties that are defined for the model. When the *Check Properties* button is pressed, all selected properties are model checked, and the results are displayed next to each property. In this case, we have not abstracted the model, so the result is very precise — a probability interval of $[0.41615, 0.41635]$[3]. The only error here is due to the termination condition of the model checker itself, and can be improved by modifying the *Transient Accuracy* field.

If we require more detailed information about the progress of the model checker, a log is available in the console view. For readability, we reproduce the output of the console, for model checking the above property, in Figure 6. In the next section, we will briefly discuss the implementation architecture in more detail, but first let us consider some results obtained by *abstracting* the active badge model — this is, after all, the purpose of our tool.

Table 1 shows the result of model checking two transient properties of the active badge model under different abstractions. The first is the same untimed until property we have been considering up to now (where we shorten *Left Corridor* to *Left*, and *Right Corridor* to *Right*). The second is a *timed* until property, which states the same condition, but with the additional constraint that the database must enter state DB_{16} (the rightmost corridor) within one time unit. For each property, we investigate the effect of aggregating the states of each of the sensors, and then finally aggregating all three of them at the same time.

The results clearly show how the choice of abstraction affects the precision of the bounds. For both properties, abstracting sensor S_{14} has no effect on the bounds — hence we can halve the size of the model without losing precision. The story for the other sensors is quite different, however. Aggregating S_{16} gives a poor bound in both cases, whereas in the case of S_{15} the bound is much worse for the first property than the second. Finally, aggregating all three sensors results in the largest reduction in the size of the model, but at the cost of limited information for the second property, and no information for the first.

We can intuitively see why aggregating S_{14} has no affect on the precision, since it cannot cause the database to move from its initial state. The other two sensors have the power to move the database to a state that satisfies the property (in the case of S_{16}), or violates the property (S_{15}), hence aggregating either

[3]We have validated our results for concrete models against PRISM.

CSL Property	Aggregated States	State Space Size	Probability Interval
$\mathcal{P}_{=?}(Left\ \mathcal{U}\ Right)$	None	72	$[0.41615, 0.41635]$
	$\{S_{14}, rep_{14}.S_{14}\}$	36	$[0.41615, 0.41635]$
	$\{S_{15}, rep_{15}.S_{15}\}$	36	$[0.06246, 1.00000]$
	$\{S_{16}, rep_{16}.S_{16}\}$	36	$[0.00000, 0.90004]$
	All of the above	9	$[0.00000, 1.00000]$
$\mathcal{P}_{=?}(Left\ \mathcal{U}^{[0,1]}\ Right)$	None	72	$[0.03018, 0.03019]$
	$\{S_{14}, rep_{14}.S_{14}\}$	36	$[0.03018, 0.03019]$
	$\{S_{15}, rep_{15}.S_{15}\}$	36	$[0.01556, 0.03199]$
	$\{S_{16}, rep_{16}.S_{16}\}$	36	$[0.00000, 0.62023]$
	All of the above	9	$[0.00000, 0.63213]$

Table 1: Abstract model checking of the active badge model

of them will have a big effect on the precision. Having said this, it is not obvious to begin with that it is safe to aggregate S_{14}, and in larger models safe abstractions can be even harder to find.

The advantage of our tool is that it allows us to experiment with different abstractions of the model, without worrying about whether or not it is safe to do so. The results of the model checker are always accurate, in that the actual probability of satisfying the property lies within the interval we obtain. Furthermore, if an abstract model satisfies or violates a particular CSL property, we can be sure that the original model also does. In the worst case, we might obtain imprecise bounds, but if we reduce the size of the model sufficiently, there is very little cost involved.

5 Architecture of the Implementation

So far, we have looked at the user interface for abstracting and model checking PEPA models without comment on the implementation details. Whilst a detailed description of the theory is beyond the scope of this paper, it is useful to know something about the type of abstraction we use, to make use of the tool more effectively[4]. A simplified view of the implementation architecture is shown in Figure 7. The dotted box contains our addition to the PEPA plug-in, and we show how it interacts with the parts of the existing tool that we take advantage of — namely, the PEPA editor and parser, and the Markov chain solvers[5].

The key to our approach is in using a *Kronecker* representation of the state space of a PEPA model [12]. Rather than deriving the state space of the system, we store a separate transition matrix for each component and action type. We can always derive the full state space by taking Kronecker sums and products of these matrices [16][6], but they give a more compact representation in general. When we abstract a PEPA model, we do so compositionally on its Kronecker state space. It is only when we come to analyse the abstract model that we have to expand the Kronecker representation. The size of this representation is proportional to the sum of the number of states in each sequential component — as opposed to the product, in the worst case, for the expanded state space.

From the Kronecker state space, we generate a graph representation of the structure of each sequential component, which we call the *display model*. This is rendered by the abstraction view, which manages a *sequential abstraction* of each component, based on the states that the user is currently aggregating. It also stores the set of atomic properties for the model, which is shared with the model checking view. The model checking view in turn keeps track of a set of CSL properties for each model.

The heart of the tool is the abstraction engine. In general, if we aggregate states in a Markov chain, we end up with a non-Markovian model, since we are no longer memoryless. However, if all the states in each aggregate partition agree on the probabilities of moving to the other aggregate partitions — a condition

[4]For a more thorough account, see [18].

[5]We use a library of solution methods provided by the Matrix Toolkits for Java (MTJ) [2]

[6]Actually, it is slightly more complicated than this — we need to keep the exit rates and transition probabilities separated, because of the minimum semantics of PEPA cooperation. For more details, see [18]

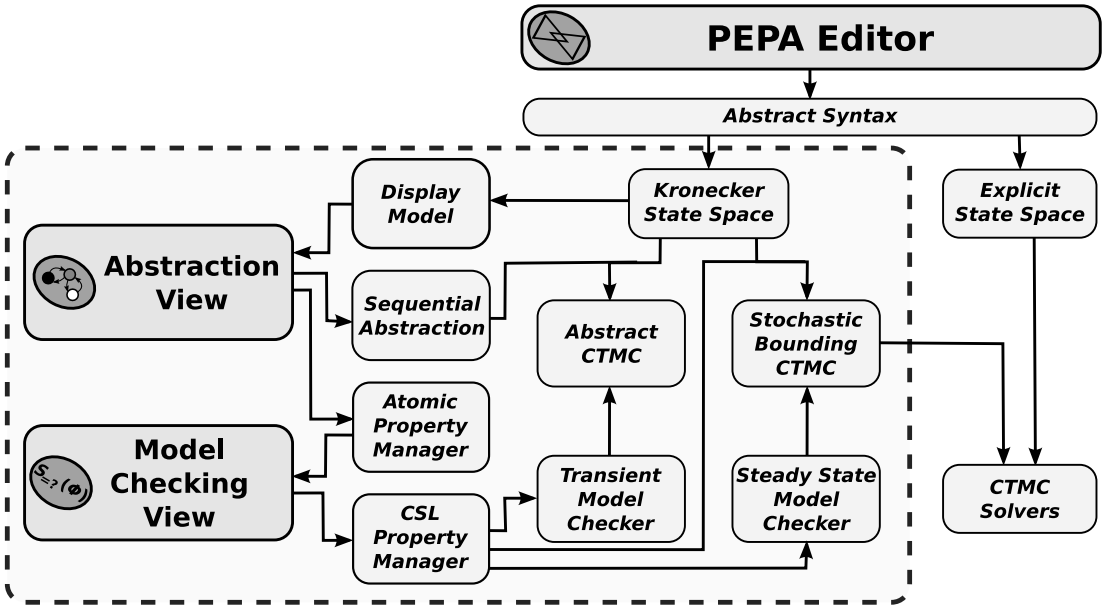


Figure 7: The architecture of the abstraction and model checking engine for PEPA

called *ordinary lumpability* [14] — then we *do* still end up with a Markov chain. Unfortunately, this condition is rare in practice, and so we need to look for alternative techniques — in our case, we make use of two:

1. An *abstract Markov chain* [7,13] is the natural result of aggregating a Markov chain that is not lumpable. Since the states in an aggregate partition have different transition probabilities, we take the maximum and minimum probability of moving between two abstract states. This is similar to a Markov decision process [17], and indeed the model checking algorithms are similar [13]. Since an abstract Markov chain is not Markovian, it has no steady state solution — hence, we cannot use it to model check steady state properties. We can, however, model check all other CSL formulae except the timed next operator[7].

   We can apply abstract Markov chains compositionally to PEPA models at a small loss of precision, by applying the abstraction before expanding the Kronecker representation. This results in an abstract CTMC that can be model checked in the usual way.

2. *Stochastic bounding* [19] of Markov chains is based on the principle of ordered probability distributions. For one Markov chain to bound the steady state distribution of another, it must ensure that its distribution is a bound at all times. For any Markov chain, given an ordering on its state space and a partitioning, we can algorithmically construct an upper (or lower) bound that is ordinarily lumpable with respect to that partitioning [8]. This can be applied compositionally to PEPA, so that the bound is constructed at the level of its Kronecker representation [18]. Since stochastic bounding produces another Markov chain, we can solve it in the usual way to compute its steady state distribution, which bounds that of the original model.

By combining these two distinct techniques, we are able to model check both transient and steady state properties of abstract PEPA models.

6 A Larger Example

Before we conclude this paper, we will examine a larger PEPA model. Figure 8 is a model of a round-robin server architecture, where the resources of a single server are shared between n computers. The server moves around each computer in turn — if there is a job waiting, it services it before moving onto the next computer.

[7]The timed next operator is not preserved under uniformisation, which is used when we apply abstract Markov chains in a continuous time setting.

$$
\begin{array}{lcl}
PC_i & = & (arrive, \lambda_i).PC'_i + (walkon_{(i+1) \bmod n}, \top).PC_i \\
PC'_i & = & (serve_i, \top).PC_i \\
Server_i & = & (walkon_{(i+1) \bmod n}, \omega).Server_{(i+1) \bmod n} + (serve_i, \mu).Server'_i \\
Server'_i & = & (walk_{(i+1) \bmod n}, \omega).Server_{(i+1) \bmod n}
\end{array}
$$

$$
(PC_0 \parallel \ldots \parallel PC_{n-1}) \underset{\{walkon_0, \ldots, walkon_{n-1}, serve_0, \ldots, serve_{n-1}\}}{\bowtie} Server_0
$$

Figure 8: A PEPA model of a round-robin server architecture

CSL Property	Aggregated States	State Space	Probability Interval
$\mathcal{S}_{=?}(Server')$	None	768	$[0.31184, 0.31184]$
	$\{Server'_{0\ldots5}\}$	7	$[0.00000, 0.33333]$
	$\{Server_{0\ldots5}\}$	7	$[0.00000, 0.75000]$
	$\{Server'_{2\ldots5}, Server_{3\ldots5}\}$	6	$[0.00000, 1.00000]$
$\mathcal{P}_{=?}(\top\ \mathcal{U}^{[0,0.1]}\ Server_2)$	None	768	$[0.53940, 0.53941]$
	$\{Server'_{0\ldots5}\}$	448	$[0.51954, 0.54567]$
	$\{Server_{0\ldots5}\}$	448	$[0.00000, 1.00000]$
	$\{Server'_{2\ldots5}, Server_{3\ldots5}\}$	384	$[0.53940, 0.53941]$

Table 2: Abstract model checking of the round-robin server model

Jobs arrive at computer PC_i at rate λ_i, and the service rate of the server is μ. The server moves between computers at rate ω.

Consider this model when $n = 6$, in which case the concrete PEPA model has 768 states. To avoid any symmetry in the model that could allow a more exact aggregation, we will assume that every computer has a different arrival rate, $\lambda_i = i + 1$. The results of model checking two distinct properties are shown in Table 2. The first property looks at the proportion of time spent in a $Server'$ state, where the server has completed a job, but has not yet moved to the next computer. The second property looks at the probability that the server will reach PC_2 within the first 0.1 time units (given that it starts with PC_0).

Looking at the steady state property first, we see that aggregating all the $Server'$ states gives a good upper bound on the actual probability. Although the lower bound provides no information, this would be a useful result if we were interested in verifying that the server spends no more than a certain proportion of time in a $Server'$ state. This is especially true when we consider that the state space has been reduced by 99%. The other choices of aggregation yield poor results, however, which illustrates how the best abstraction depends entirely on the property we are analysing.

If we look at the second property, by comparison, we see that we achieve the best results when we abstract all the states following $Server_2$, but before $Server_0$. In this case, we can halve the state space without affecting the precision. In fact, since Table 2 only looks at aggregating states on the server, we can do even better. If we aggregate the computer states for $PC_2 \ldots PC_5$, we can reduce the state space to just 24 states (a 97% reduction in size) without affecting the precision. The reason we can achieve such good results here, is that after the server passes through the $Server_2$ state, the property must be satisfied — hence we can ignore all subsequent states. The abstraction view allows us to take advantage of this aggregation very quickly, without requiring any modifications to the model.

7 Conclusions

In this paper, we have described a new tool for abstracting and model checking PEPA models, which is an extension of the Eclipse PEPA plug-in. It provides a graphical interface for labelling and aggregating states of PEPA components, and for constructing and model checking CSL properties. The key advantage of the

tool is that it allows modellers to quickly experiment with different ways of abstracting their models, and to take advantage of direct model checking facilities in the PEPA plug-in, without requiring external tools such as PRISM. This is not to say that our tool is a replacement for other, more established model checkers, but we feel that it is a useful addition to the artillery of the performance modeller.

Whilst a great deal has gone into the development of this tool, we are aware that there are still many areas in which its capabilities can be expanded upon and improved. For example, we would like to add support in the near future for PEPA's aggregation notation, which provides a shorthand for specifying a number of identical copies of the same component (e.g. *Client*[100]). We also intend to add support for exporting and importing of CSL properties, so that the plug-in can more easily be used in conjunction with PRISM and other tools.

Overall, whilst we have demonstrated the capabilities of our tool in this paper, our ultimate objective is to provide a practical and useful contribution to the model checking community. We would be delighted if you, the reader, could take the time to download and experiment with the new PEPA plug-in, and to provide us with any feedback, comments, or suggestions for improvement. It is easy to install, and full instructions are given at `http://www.dcs.ed.ac.uk/pepa/tools/plugin`. May many happy abstractions await you!

References

[1] The Eclipse platform. `http://www.eclipse.org`.

[2] Matrix Toolkits for Java (MTJ). `http://code.google.com/p/matrix-toolkits-java/`.

[3] A. Aziz, K. Sanwal, V. Singhal, and R. Brayton. Verifying continuous time Markov chains. In *Computer Aided Verification (CAV)*, number 1102 in Lecture Notes in Computer Science, pages 269–276. Springer-Verlag, 1996.

[4] C. Baier, B.R. Haverkort, H. Hermanns, and J-P. Katoen. Model checking continuous-time Markov chains by transient analysis. In *Computer Aided Verification (CAV)*, pages 358–372, 2000.

[5] J. T. Bradley and W.J. Knottenbelt. The ipc/HYDRA tool chain for the analysis of PEPA models. In *QEST '04: Proceedings of the The Quantitative Evaluation of Systems, First International Conference*, pages 334–335, Washington, DC, USA, 2004. IEEE Computer Society Press.

[6] G. Clark, S. Gilmore, and J. Hillston. Specifying performance measures for PEPA. In *ARTS '99: Proceedings of the 5th International AMAST Workshop on Formal Methods for Real-Time and Probabilistic Systems*, pages 211–227, London, UK, 1999. Springer-Verlag.

[7] H. Fecher, M. Leucker, and V. Wolf. Don't know in probabilistic systems. In *Proceedings of SPIN'06*, number 3925 in Lecture Notes in Computer Science, pages 71–88, 2006.

[8] J-M. Fourneau, M. Lecoz, and F. Quessette. Algorithms for an irreducible and lumpable strong stochastic bound. *Linear Algebra and its Applications*, 386:167–185, 2004.

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

[10] J. Hillston. *A Compositional Approach to Performance Modelling*. Cambridge University Press, 1996.

[11] J. Hillston. Fluid flow approximation of PEPA models. In *Proceedings of the Second International Conference on the Quantitative Evaluation of Systems*, pages 33–43, Torino, Italy, Sep 2005. IEEE Computer Society Press.

[12] J. Hillston and L. Kloul. An efficient Kronecker representation for PEPA models. In *Proceedings of the Joint International Workshop, PAPM-PROBMIV 2001*, number 2165 in Lecture Notes in Computer Science, pages 120–135. Springer-Verlag, 2001.

[13] J-P. Katoen, D. Klink, M. Leucker, and V. Wolf. Three-valued abstraction for continuous-time Markov chains. In W. Damm and H. Herrmanns, editors, *Proceedings of 19th International Conference on Computer-Aided Verification (CAV'07)*, number 4590 in Lecture Notes in Computer Science, pages 316–329. Springer-Verlag, 2007.

[14] J.G. Kemeny and J.L. Snell. *Finite Markov Chains*. Springer-Verlag, 1976.

[15] M.Z. Kwiatkowska, G. Norman, and D. Parker. PRISM: Probabilistic symbolic model checker. In *Computer Performance Evaluation: Modelling Techniques and Tools*, number 2324 in Lecture Notes in Computer Science, pages 200–204, 2002.

[16] B. Plateau. On the stochastic structure of parallelism and synchronization models for distributed algorithms. *SIGMETRICS Performance Evaluation Review*, 13(2):147–154, 1985.

[17] M.L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, NY, USA, 1994.

[18] M.J.A. Smith. Compositional abstraction of PEPA models, 2009. In submission. Available from: `http://lanther.co.uk/papers/PEPA_abstraction.pdf`.

[19] D. Stoyan. *Comparison Methods for Queues and Other Stochastic Models*. Wiley & Sons, New York, NY, USA, 1983.

[20] M. Tribastone. The PEPA plug-in project. In M. Harchol-Balter, M. Kwiatkowska, and M. Telek, editors, *Proceedings of the 4th International Conference on the Quantitative Evaluation of SysTems (QEST)*, pages 53–54. IEEE Computer Society Press, 2007.

Concerning Performance Driven Cryptographic Protocol Development

Nicholas O'Shea

Abstract

We present a method of applying performance modelling to cryptographic protocols and explore how this allows protocol developers to determine if there are any bottlenecks in their protocol. When the deployment scenario has been set implementers can determine how a chosen protocol works in that scenario. Using the tools provided by the PAM framework network administrators can also determine the most efficient use of any upgrade to the network infrastructure.

1 Introduction and Motivation

Since the dawn of civilisation, man has felt an inescapable need to communicate securely. While we can now provide this, performance is looked at as a critical matter. Noted security expert Bruce Schneier repeatedly states in [5] that "Security is a trade-off". While people are willing to wait for security, there are limits to how long they will do so. There are multiple methods for analysing the security properties of cryptographic protocols and equally many methods of analysing the performance of computer systems. In this paper we explore merging the two fields.

2 PAMeLa

PAM, Process Algebra Modelling, is a framework for representing process calculi as labelled continuous-time Markov chains and analysing them using the analysers of the PEPA Eclipse Plug-in[6]. PAM allows stakeholders in Markovian process calculi to generate the underlying labelled transition system from their favourite Markovian process calculus and then pass this transition system to the PEPA Eclipse Plug-in for solution and visualisation of the results. The PAM language represents transition systems using XML allowing the structure of the transition system to be easily represented.

LySa[2] is a process calculus for describing cryptographic protocols similar to the Spi-calculus[1]. Using static analysis there are tools for analysing the security properties of protocols modelled in this language.

We present PAMeLa, short for PAM-ersatz-LySa, an Eclipse plug-in which translates LySa models into PAM files which we can then analyse. The translation process between LySa and PAM is straightforward for simple protocols but as the protocols get larger subtleties are discovered. A PAM file contains a series of states which can have several transitions. A transition is a description of the resulting state along with a label and rate for the transition from the original to the new state. For our use, a state is a

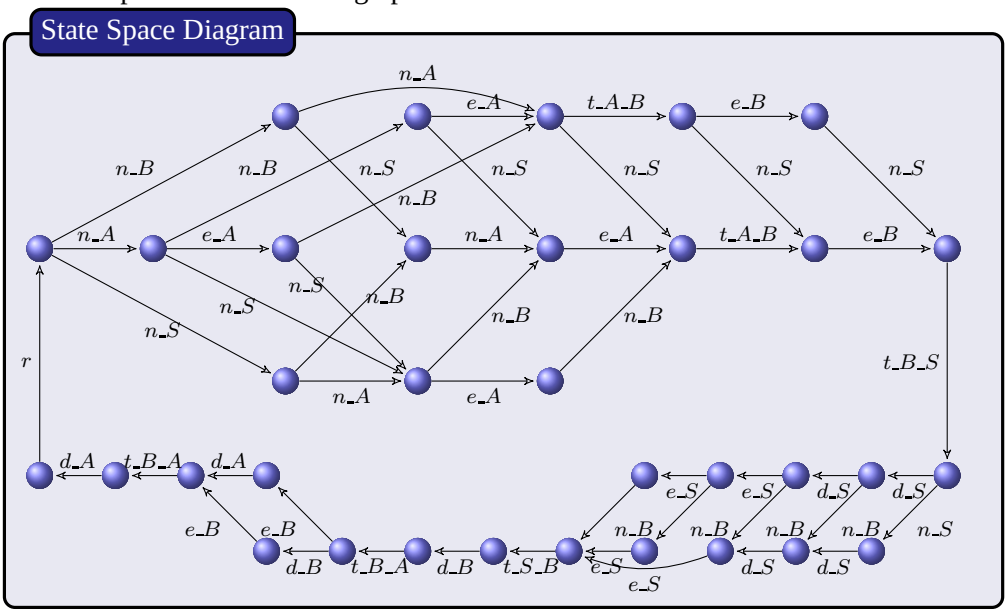
composition with each process representing a principal in a protocol. As cryptographic protocols do not typically have a notion of choice the transitions in a single process are linear. There is some work to be done with automatically unrolling multiple and nested encryption. Additionally, although there is no choice in a single principal there is some option in the transitions. In the excerpt below, for example, either the first or third process can transition to a new state, although the second is waiting for an accompanying send process from principal B for this receive process.

```
<state>
 <composition>
  <process>(new NA)</process>
  <process>(new NB)</process>
  <process>(B,S,M,A,B;z1,z2)</process>
 </composition>
 <transitions>
  <transition>
   <composition>
    <process>(A,B,M,A,B,(A,B,M,NA))</process>
    <process>(new NB)</process>
    <process>(B,S,M,A,B;z1,z2)</process>
   </composition>
   <via name="(new NA (by A))" rate="n_A"/>
  </transition>
  <transition>
   <composition>
    <process>(new NA)</process>
    <process>(A,B,M,A,B;y1)</process>
    <process>(B,S,M,A,B;z1,z2)</process>
   </composition>
   <via name="(new NB (by B))" rate="n_B"/>
  </transition>
 </transitions>
</state>
```

This section comes from the Otway-Rees protocol presented below and the current stage this section represents is emphasised.
(ν KAS) (ν KBS)
(!**(ν NA)** $\langle$A,B,M,A,B,$\{$A,B,M,NA$\}_{KAS}$ [at a1 dest { s1 }] $\rangle$.
(B,A,M;x1). decrypt x1 as $\{$NA;xk$\}_{KAS}$ [at a2 orig { s3 }] in
(B,A;x2). decrypt x2 as $\{$;xmsg$\}_{xk}$ [at a3 orig { b3 }] in 0
|
!**(ν NB)** (A,B,M,A,B;y1).
$\langle$B,S,M,A,B,y1,$\{$A,B,M,NB$\}_{KBS}$ [at b1 dest { s2 }] $\rangle$.
(ν MSG) (S,B,M;y2,y3).
decrypt y3 as $\{$NB;yk$\}_{KBS}$ [at b2 orig { s4 }] in
$\langle$B,A,M,y2$\rangle$.$\langle$B,A,$\{$MSG$\}_{yk}$ [at b3 dest { a3 }] $\rangle$.0
|
!(ν K) **(B,S,M,A,B;z1,z2)**.
decrypt z1 as $\{$A,B,M;zna$\}_{KAS}$ [at s1 orig { a1 }] in
decrypt z2 as $\{$A,B,M;znb$\}_{KBS}$ [at s2 orig { b1 }] in
$\langle$S,B,M,$\{$zna,K$\}_{KAS}$ [at s3 dest $\{$a2$\}$], $\{$znb,K$\}_{KBS}$ [at s4 dest $\{$b2$\}$] $\rangle$.0)

The full state space diagram can be seen below, such diagrams can be automatically generated by the PAMeLa Eclipse plug-in, although this one has been hand drawn in an attempt to provide readability. The diagram demonstrates that while each principal

is a linear system, there are several areas where different principals can advance at different speeds before meeting up on shared transitions.



Different rates are generated for encryption, decryption, message generation and communication for each separate principal and link between them. Using the PEPA Eclipse Plug-in we can then determine if the protocol is suitable for the intended deployment scenario and if we were determined to improve one part of the infrastructure which part it should be. In cases such as the above Otway-Rees protocol, preconditions on the protocol such as the shared keys KAS, KBS are removed from the PAM model.

3 AutoRate

Without specifying the rates of the various transitions there is only so much benefit that can be gained from analysis of the model. By declaring the type of connections and devices we can automatically assign values to these required rates. This is done with a graphical interface that makes it easier for those unfamiliar with the formal model to achieve desirable results quickly. A screen-shot of this can be seen in Figure 1. A user can choose the type of device that the principal will run on and the network speed of the connection between communicating principals. This information will then be used to automatically calculate the rates for all the transitions. This particular deployment scenario automatically generates the following rates:

```
<parameter name="e_S" value="8.961646000000002"/>
<parameter name="t_B_A" value="135.0"/>
<parameter name="n_S" value="2330.0"/>
<parameter name="t_S_B" value="250.0"/>
<parameter name="d_B" value="1165.0"/>
<parameter name="d_A" value="550.0"/>
<parameter name="n_B" value="2330.0"/>
<parameter name="n_A" value="1100.0"/>
<parameter name="t_B_S" value="250.0"/>
<parameter name="e_B" value="8.961646000000002"/>
<parameter name="d_S" value="1165.0"/>
```

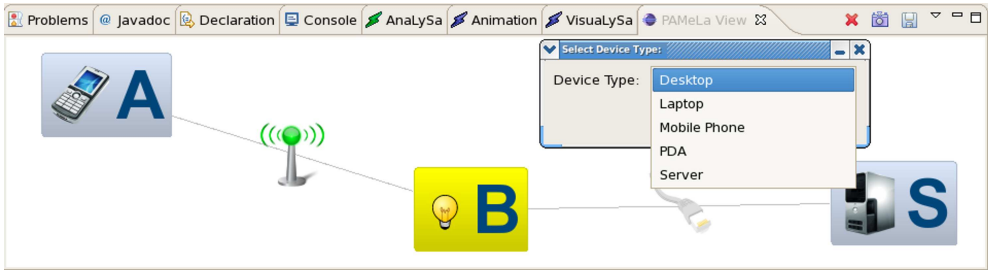


Figure 1: AutoRate Screenshot

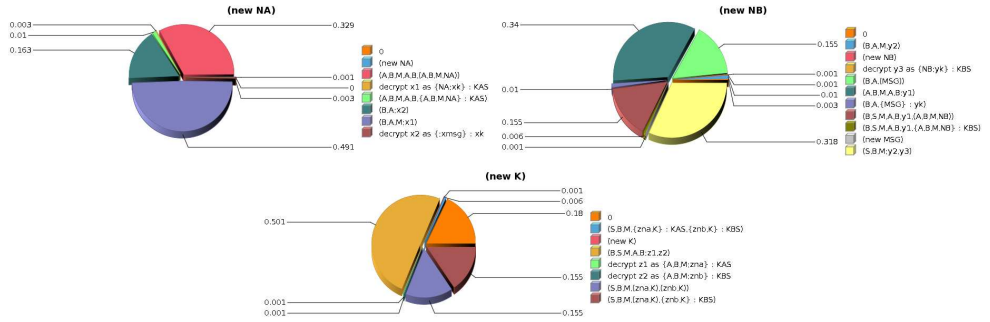


Figure 2: Utilisation Graphs of principals (clockwise from top left) A, B and S

```
<parameter name="r" value="6.2E9"/>
<parameter name="t_A_B" value="135.0"/>
<parameter name="e_A" value="4.2308200000000005"/>
```

4 Analysing Otway-Rees

If we look at the graphs presented in Figure 2 for each protocol's utilisation we see that most of the time is spent waiting for a message to arrive. This tells us that the devices that are used are sufficient to not be a bottleneck on the system. Looking at experimentation graphs we see that there is a certain point where it is worth upgrading this component after which improving the speed of the network will have little effect. Figure 3 is typical of all the transmission speed experimentation graphs and we can see the curve levels off at a rate of around 80 and anything after that makes little difference. As our rates are over this threshold while we can improve the performance it would not be an efficient use of resources to do so as the performance increase would be marginal without a large investment.

5 Conclusion

It is important for implementers to be able to make sure that they choose an appropriate protocol for their deployment scenario. For protocol developers it is equally important that their design is not so flawed as to include unnecessary performance bottlenecks.

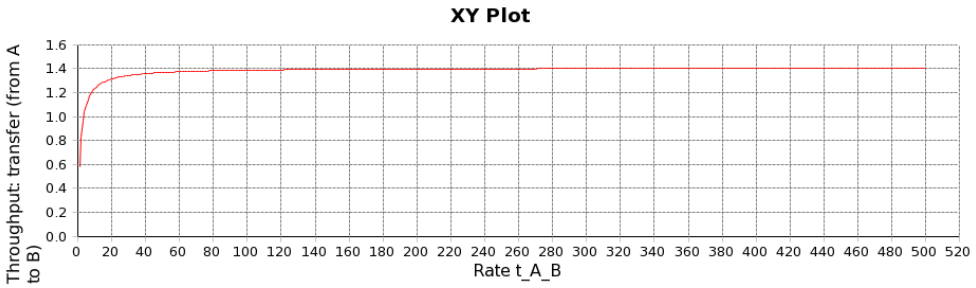


Figure 3: Varying the rate of transmission from principal A to principal B

PAMeLa allows protocol developers to utilise powerful performance analysis without needing any knowledge of the underlying mechanics or language. With the aid of the AutoRate plug-in appropriate rates are automatically calculated based on a user's selected deployment scenario, eliminating a difficult aspect of performance modelling. With these tools, network administrators can work out the best way of improving the performance of their network thus potentially saving time and money instead of accidentally devoting resources to areas which would not improve performance.

A theoretical approach to performance evaluation for security protocols has already been attempted in work such as [3]. This work relies on establishing a new extended operational semantic for LySa in which each transition is assigned a label. Here we provide an easy-to-use alternative that fits in with both the PEPA Eclipse Plug-in and the LySa Toolkit in Eclipse (LyTE)[4].

There are a few avenues for further work. It would be useful to be able to perform response time analysis on the generated PAM models. The rates for communication and encryption could be made size-dependant. Additionally more simulation work to improve the pre-defined rates would be beneficial, perhaps providing different rates for different encryption algorithms or for devices under different loads with multiple instances of a protocol running.

References

[1] Martín Abadi and Andrew D. Gordon. A Calculus for Cryptographic Protocols: The Spi Calculus. In *Fourth ACM Conference on Computer and Communications Security*, pages 36–47. ACM Press, 1997.

[2] C. Bodei, M. Buchholtz, P. Degano, F. Nielson, and H. Nielson. Automatic validation of protocol narration. *Proceedings of 16th IEEE Computer Security Foundations Workshop (CSFW 16)*, pages 126–140, 2003.

[3] Chiara Bodei, Michele Curti, Pierpaolo Degano, Mikael Buchholtz, Flemming Nielson, Hanne Riis Nielson, and Corrado Priami. Performance Evaluation of Security Protocols Specified in LySa. *Electr. Notes Theor. Comput. Sci.*, 112:167–189, 2005.

[4] Nicholas O'Shea. Protocol Analysis in a new LyTE. In *Proceedings of The 13th Nordic Workshop on Secure IT Systems*, pages 83–94, 2008.

[5] Bruce Schneier. *Secrets and Lies : Digital Security in a Networked World*. Wiley, January 2004.

[6] M. Tribastone, A. Duguid, and S. Gilmore. The PEPA Eclipse Plug-in. *Performance Evaluation Review*, 36(4):28–33, March 2009.

Part II

Bio-PASTA

A computational method based on temporal logic for parameter search and robustness analysis of biological models — Invited Talk —

François Fages
INRIA Paris-Rocquencourt, France

Abstract

Temporal logics have proven useful as specification languages for describing the behavior of a broad variety of systems ranging from electronic circuits to software programs, and more recently biological systems in either boolean, discrete, stochastic or continuous settings. Because temporal logics allow us to express both qualitative (e.g. some protein is eventually produced) and quantitative (e.g. a concentration exceeds 10) information about time and systems variables, they provide a powerful specification language in comparison with the essentially qualitative properties considered in dynamical systems theory (e.g. multistability, existence of oscillations) or with the exact quantitative properties considered in optimization theory (e.g. curve fitting). In this talk, we define a continuous degree of satisfaction of a temporal logic formula with constraints, and show how it can be used as a fitness function with state-of-the-art optimization methods for finding kinetic parameter values satisfying a set of biological properties formalized in temporal logic. We also show how it can be used to define a measure of robustness of a biological model with respect to some temporal specification. These methods, implemented in BIOCHAM, are evaluated on models of the cell cycle and of the MAPK signalling cascade.

(Joint work with Aurélien Rizk, Grégory Batt and Sylvain Soliman)

Modelling Scaffold-mediated Crosstalk between the cAMP and the Raf-1/MEK/ERK Pathways

Oana Andrei and Muffy Calder

Department of Computing Science, University of Glasgow
Glasgow G12 8RZ, UK

Abstract. We study the biochemical processes involved in scaffold-mediated crosstalk between the cAMP and the Raf-1/MEK/ERK pathways. We model the interactions by a continuous time Markov chain with levels and analyse properties using Continuous Stochastic Logic and the symbolic probabilistic model checker PRISM. We consider a number of biologically relevant properties of the model, including sequentially dependent events and pulsating behaviour. In order to handle these kinds of properties, we extend the model with signs of first order derivatives.

1 Introduction

In the context of intracellular signalling pathways, a scaffold is a protein whose main role is to anchor particular proteins in the proper locations for receiving signals or transmitting them. In addition, under certain circumstances, a scaffold can increase the output of a signalling cascade or decrease the response time for a faster output.

In this paper we present a model of scaffold-mediated crosstalk between the cyclic adenosine monophosphate (cAMP) and the Raf-1/MEK/ERK pathways, an interaction that has an important role in the regulation of cell proliferation, transformation and survival. We use continuous time Markov chains with levels (CTMC with levels) [CDHC09] and the PRISM model checker [KNP07] to develop and analyse a number of predictive models. Our aim is to provide new quantitative analysis and new ways to model and reason about behaviour that is not yet completely understood or quantified.

We express properties in Continuous Stochastic Logic (CSL) [BHHK03], in particular some restrictive case of sequentiality properties and pulsations. In order to analyse these kinds of properties, for each variable of interest we add a new variable in the model representing the sign of its derivative.

The paper is organised as follows. In the next section we introduce the basics about scaffolds and the role of an AKAP scaffold in particular. In Section 3 we give an overview of the PRISM model. Section 4 contains a description of some properties of interest and the results of analysis. Conclusion and directions for future work follow. We note that the behaviour of scaffold proteins modelled here is the topic of current wet-lab investigation. Our models have been informed by discussions with a number of experimentalists: some aspects of behaviour are still unknown and the subject of conjecture.

2 Scaffold proteins and the AKAP

In intracellular signal transduction pathways, scaffolds are proteins playing an organisational role rather than a signalling role [JEF00]. Scaffolds have a anchoring function

by placing particular proteins in the proper intracellular locations for receiving signals or transmitting them. Experiments have shown that under certain circumstances, a kinase scaffold can increase the output of a signalling cascade or decrease the response time for a faster output. Therefore scaffolds may also exhibit a catalytic function. Another property of the scaffolds is combinatorial inhibition: if there are too many or too few of either scaffolds or kinases, the output of the pathway decreases.

We are interested in particular in the behaviour of the *A-kinase anchoring protein* (AKAP for short) in the context of crosstalk between cyclic AMP and the Raf-1/MEK/ERK pathway. The species we are interested in are the following: (i) cyclic adenosine monophosphate (cAMP); (ii) protein kinase A (PKA), the main cAMP effector; (iii) Raf-1 with two phosphorylation sites of interest, Serine 338 (S338) and Serine 259 (S259); (iv) phosphodiesterase 8 (PDE8A1), (v) phosphatase PP.

If the concentration level of cAMP goes above the basal one, cAMP activates PKA by binding to its regulatory subunits. When PKA becomes activate, its catalytic subunits catalyse the transfer of ATP terminal phosphates to the phosphorylation site S259 of Raf-1. The site S338 of Raf-1 is inhibited when S259 is phosphorylated. Only when S338 gets phosphorylated, the pathway Raf-1/MEK/ERK is activated and the signalling cascade begins.

The catalytic function of PKA would sometimes couple with the AKAP, by binding PKA together with phosphodiesterase PDE8A1 on the scaffold to form a complex that functions as a signal module. Under these conditions, as the cell is stimulated, cAMP activates PKA, and then PKA is responsible for the activation of PDE8A1 (by phosphorylation). PDE8A1 converts cAMP to AMP by hydrolysis. If phosphorylated, PDE8A1 degrades more cAMP, hence rapidly reducing the amount of cAMP that can activate PKA hence leading to a feedback mechanism for downregulating PKA.

The inhibition of Raf-1 at S338 is correlated with a high activity of PKA. At the beginning, cAMP synthesis is induced, causing a rise of PKA's activity that continues in the inhibition of Raf-1.

We illustrate the AKAP scaffold and the regulatory mechanism described above in Figure 1. We use three different types of arrow to distinguish between different types of interactions:

- A activates or phosphorylates B : $A \longrightarrow\!\!\succ B$
- A dephosphorylates B : $A - - \succ B$
- A degrades B : $A \longrightarrow\!\!\dashv B$

The arrow with no source and with target cAMP represents a diffusion of cAMP from the environment.

The AKAP scaffold has three positions to be filled by PKA, Raf-1 and PDE8A1 respectively. Hereafter we use a binary representation of the states of each position: 1 for activation or phosphorylation and 0 otherwise. The second position concerns the state of the site S259 of Raf-1. If the scaffold position for PDE8A1 is not filled, we only represent the first two positions of the scaffold. For instance $S100$ stands for a filled scaffold with active PKA and unphosphorylated PDE8A1 and site S259, whereas $S01$ for an unfilled scaffolded with inactive PKA and phosphorylated S259.

In Figure 2 we describe the biochemical reactions of the model. Each reaction is given in pseudo-chemical notation, with explicit reference to the scaffold positions (the underlying reactions have mass action kinetics). We associate reaction rate constants (from r_1 to r_{26}) with each biochemical reaction.

Currently, we do not have good experimental data concerning rates for the reactions. However, we have some information on the ratio between the rate of PKA

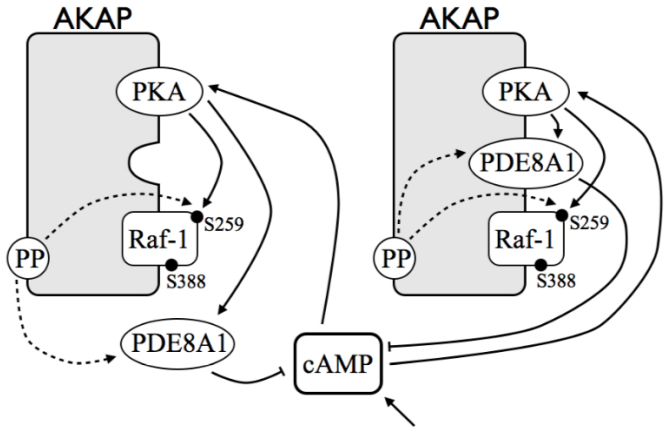


Fig. 1. Interactions between cAMP, filled AKAP scaffold, unfilled scaffold and free PDE8A1

(cAMP diffusion)
$\_ \rightarrow_{r_1} cAMP$

(PKA activation)
$S000 + cAMP \rightarrow_{r_2} S100$
$S00 + cAMP \rightarrow_{r_3} S10$

(S259 phosphorylation)
$S100 \rightarrow_{r_4} S110$
$S101 \rightarrow_{r_5} S111$
$S10 \rightarrow_{r_6} S11$

(S259 dephosphorylation)
$PP + S010 \rightarrow_{r_7} PP + S000$
$PP + S011 \rightarrow_{r_8} PP + S001$
$PP + S01 \rightarrow_{r_9} PP + S00$

(cAMP release)
$S111 \rightarrow_{r_{17}} S011 + cAMP$
$S11 \rightarrow_{r_{18}} S01 + cAMP$

(PDE8A1 phosphorylation)
$S100 \rightarrow_{r_{10}} S101$
$S110 \rightarrow_{r_{11}} S111$
$S10 + PDE8A1 \rightarrow_{r_{12}} S10 + pPDE8A1$
$S11 + PDE8A1 \rightarrow_{r_{13}} S11 + pPDE8A1$

(PDE8A1 dephosphorylation)
$PP + S001 \rightarrow_{r_{14}} PP + S000$
$PP + S011 \rightarrow_{r_{15}} PP + S010$
$PP + pPDE8A1 \rightarrow_{r_{16}} PP + PDE8A1$

(cAMP degradation)
$S011 + cAMP \rightarrow_{r_{19}} S011$
$S001 + cAMP \rightarrow_{r_{20}} S001$
$S100 + cAMP \rightarrow_{r_{21}} S100$
$S110 + cAMP \rightarrow_{r_{22}} S110$
$S010 + cAMP \rightarrow_{r_{23}} S010$
$S000 + cAMP \rightarrow_{r_{24}} S000$
$pPDE8A1 + cAMP \rightarrow_{r_{25}} pPDE8A1$
$PDE8A1 + cAMP \rightarrow_{r_{26}} PDE8A1$

Fig. 2. Biochemical reactions occurring during scaffold-mediated crosstalk between the cAMP and the Raf-1/MEK/ERK pathway

phosphorylating Raf-1 at S259 and PDE8A1 (either on the scaffold or not). On unfilled scaffolds, PKA phosphorylates two or three times less unscaffolded PDE8A1 than Raf-1 at S259 from the same scaffold. On filled scaffolds, PKA phosphorylates at the same rate Raf-1 at S259 and PDE8A1. Consequently the relation between constant rates of the reactions involving PKA phosphorylating either PDE8A1 or Raf-1 is: $r_4 = r_5 = r_6 = r_{10} = r_{11} = 3 * r_{12} = 3 * r_{13}$. In addition, phosphorylated PDE8A1 degrades about three times more cAMP than PDE8A1 does, hence the following ratios between the constants rates of the reactions where PDE8A1 degrades cAMP: $r_{19} = r_{20} = r_{21} = r_{22} = r_{23} = r_{24} = 3 * r_{25} = 9 * r_{26}$.

Finally, when PKA and PDE8A1 form a complex on the scaffold, PKA's activity becomes more efficient.

Our discussions with experimentalists have revealed the following expectations, or conjectures, about AKAP behaviour.

Question 1. Increasing the amount of phosphorylated PDE8A1 leads to a cascade of changes in the concentration levels of the other reactants: decreasing amounts of cAMP and active PKA, and an increase in the activity of Raf-1 due to lower levels of phosphorylated Raf-1 at site S259. Informally, we express this behaviour by the following implication, where $\uparrow$ (resp. $\downarrow$) denotes increase (resp. decrease):

$$\uparrow \text{pPDE8A1} \Rightarrow \downarrow \text{cAMP} \Rightarrow \downarrow \text{active PKA} \Rightarrow \downarrow \text{pRaf-1}_{\text{S259}} \equiv \uparrow \text{active Raf-1}$$

Question 2. Interactions between cAMP and Raf-1 and other molecules are not considered in the current model. However, the system is not closed, we include a kind of exogenous interaction concerning the diffusion of cAMP. We conjecture this leads to a fluctuation of the concentrations of some reactants. Namely, the system should exhibit a pulsating behaviour corresponding to the feedback mechanism for the downregulation of PKA coupled with the diffusion of cAMP. Time courses from laboratory experiments suggest the presence of a pulsating behaviour. Pulsation ensures that the state of the Raf-1 pathway alternates such that it is not always (or for a very long period of time) either active or inactive (which may increase the risk of disease). Note that we call such a behaviour pulsating, not oscillating. This is because oscillation assumes fluctuation around a given value; current data does not provide us with such a value, hence our choice of pulsating rather than oscillating behaviour.

3 Modelling AKAP with CTMCs with Levels

Following the style adopted in [CVGO06], we define a PRISM model for a CTMC with levels as follows. There are modules for cAMP, scaffold, PDE8A1 and PP, each with corresponding variables representing levels of concentrations. In particular, the module for the scaffold has a variable for each possible combination of positions (S000, S100, S101, S110, S011, S010, S001, S111, S00, S10, S01, S11). Commands in the modules correspond to reactions, which are synchronised on each participating module (i.e. consumers and producers in the chemical reaction). Additionally, we defined diffusion of cAMP from time to time.

As an example, consider the reaction r_2 where cAMP activates PKA when the level of cAMP is above the basal level. Then in the module describing cAMP we add the command:

```
[activate_PKA] (cAMP > basal_camp) -> (cAMP) : (cAMP' = cAMP-1);
```

while in the module describing the scaffold we have the coupling command:

```
[activate_PKA] (S000 > 0) & (S100 < scaffold_max) ->
               (r2*S000) : (S100' = S100+1) & (S000' = S000-1);
```

In the definition of the PRISM model, we used the convention that the rate constant goes in the command corresponding to the consumer reactant, in this case PKA. By synchronisation on the common label, the reaction rate will then be the product of the constant rate r_2 and the concentration levels of cAMP and PKA.

Unless otherwise stated, we assume the number of levels $N = 2$. We consider maximum N levels of filled/unfilled scaffolds, $2 * N$ levels of phosphatase PP, and around $N/2$ levels of unscaffolded PDE8A1. Whereas for cAMP, since it is diffused in the system, we allow a greater concentration of cAMP, maximum $10 * N$. Full details of the model are available from the authors.

4 Analysis

We use rewards based properties and CSL properties to formalise the questions posed in Section 2 and PRISM to verify their satisfaction (or not).

First, we consider Question 1, using rewards to compute the expected level of concentration at a particular time.

Second, we consider transient properties concerning the sequentiality of events and pulsating behaviour. We extend the states of the CTMC model to include (signs of) derivatives, for some variables of interest.

Reward-based Properties. For each species of interest: phosphorylated PDE8A1, free cAMP (not bound to some PKA), active PKA and phosphorylated Raf-1 at site S259, we add a reward construct to compute the expected level of concentration at a particular time. For example, the reward associated with phosphorylated PDE8A1 computes the sum of the level of unscaffolded and phosphorylated PDE8A1 and the levels of scaffolded and phosphorylated PDE8A1, at each time instant.

```
rewards "phosphopde8"
    true : pPDE8A1 + S101 + S111 + S001 + S011;
endrewards
```

In Figure 3 we plot the expected levels of concentration for each species. Note a delayed pulsation of all variable values. However, this does not prove that the properties expressed in Section 2 are satisfied; for this, we formulate temporal properties.

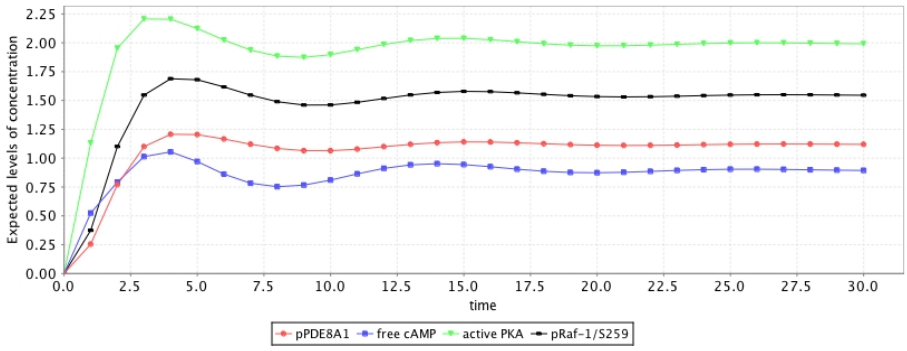


Fig. 3. Expected levels of concentrations of phosphorylated PDE8A1, free cAMP, active PKA, and phosphorylated Raf-1 at site S259 after 30 time-units

Derivative-based Transient Properties. We extend the CTMC with levels model such that for each species in the biochemical system, we add a new variable representing the sign of the value of the derivative. Then, each PRISM command associated with a transition updates not only the values of some variables, but also the signs of their derivative as well. For efficiency reasons we include a derivative only for variables occurring in the properties of interest. In our particular model, we add derivatives for cAMP, active PKA and scaffolded phosphorylated PDE8A1.

We illustrate here how the signs of the derivatives are updated in the scaffold module for the reaction where cAMP activates PKA:

```
[activate_PKA] (S000 > 0) & (S100 < scaffold_max) ->
               (r2*S000) : (S100' = S100+1) & (S000' = S000-1) &
               (drv_PKA_A' = 1) & (drv_S259_P' = 0) & (drv_PDE8_P' = 0);
```

Since the level of active PKA is affected (increased) by this command, the derivative of PKA becomes positive, whereas the rest of derivatives become 0 since there is no change in the variables values.

We now formalise the temporal properties. In the following, for x a variable, we denote by $\downarrow x$ a negative derivative of x and by $\uparrow x$ a positive derivative of x.

Necessarily Preceded. We express the property in Question 1 as a temporal query using the *necessarily preceded* or *requirement* pattern [MRM+08]. This pattern represents an ordering relation between two events, the occurrence of the later being conditioned by the occurrence of the former: *a state ϕ is reachable and is necessary preceded all the time by a state ψ*. The associated CTL formula of this pattern is:

$$EF\phi \wedge (AG((\neg\psi) \Rightarrow AG(\neg\phi)))$$

Assume the following two state formula:

$$\phi = \downarrow \text{cAMP} \wedge \downarrow \text{active PKA} \qquad \psi = \uparrow \text{pPDE8A1}$$

with ϕ corresponding to a state where the levels of cAMP and active PKA are decreasing, and ψ to a state where the level of phosphorylated PDE8A1 is increasing. Employing basic propositions equivalences, we translate the requirement pattern into CSL to obtain the following formula which was checked as **true** for our PRISM model:

$$P_{>0}[F\phi] \wedge P_{\leq 0}[F(\neg((\neg\psi) \Rightarrow P_{\geq 1}[F(\neg\phi)]))]$$

Pulsating behaviour. An oscillating behaviour concerns fluctuation around a given value k. Oscillation and its expression as temporal formulas in CTL and PCTL have been studied in [BMM09] and informally described as *always in the future, the variable x departs from and reaches the values k infinitely often.* The corresponding CTL formula is:

$$AG(((x = k) \Rightarrow EF(x \neq k)) \wedge ((x \neq k) \Rightarrow EF(x = k)))$$

However, as discussed earlier, we are interested in pulsating behaviour, i.e. no fixed k. We therefore consider oscillations (around 0) of the values of the first derivatives of some variables. We refer to this approximate oscillating behaviour as pulsation. Note that we can observe a pattern corresponding to a pulsation in Figure 3: we repeatedly have the situation where the level of phosphorylated PDE8A1 increased whereas the levels of cAMP and active PKA decreased, and then the level of phosphorylated

PDE8A1 decreased whereas the levels of cAMP and active PKA increased. Assume the following two state formulas:

$$\phi = \uparrow \text{pPDE8A1} \wedge \downarrow \text{cAMP} \wedge \downarrow \text{active PKA}$$
$$\psi = \downarrow \text{pPDE8A1} \wedge \uparrow \text{cAMP} \wedge \uparrow \text{active PKA}$$

We translate the CTL temporal formula describing an oscillation to a CSL formula for a pulsation involving the two state formulas above and we obtain the following formula checked as **true** for our model using PRISM:

$$P_{\leq 0}[F(\neg(\phi \Rightarrow P_{>0}[F\psi]) \vee \neg(\psi \Rightarrow P_{>0}[F\phi]))]$$

5 Related work

Derivatives have been considered previously in the context of model checking. For example in BIOCHAM [Fag05,RBFS08], the query language associated is LTL with constraints over real numbers, evaluated using a symbolic model checker written in Prolog. Again, in the context of BIOCHAM [CRCFS04], a weaker form of oscillation properties expressed in CTL are used with the symbolic model checker NuSMV; the oscillating behaviour is approximated by the necessary but not sufficient formula $EG((EF\neg\varphi) \wedge (EF\varphi))$.

6 Conclusion and Future Work

We have developed a stochastic model of the behaviour of the AKAP scaffold and scaffold-mediated crosstalk between the cAMP and the Raf-1/MEK/ERK pathways. The model is a CTMC with levels and is implemented in the PRISM language.

We have considered questions and conjectures concerning system behaviour posed by experimentalists; these include sequentially dependent events and pulsating behaviour. In the context of imprecise and incomplete data, pulsation seems more appropriate than oscillation. We have used rewards and CSL to express the properties, and checked them with the PRISM model checker. In order to express pulsation, we have added a representation of signs of first derivatives to the stochastic model. Preliminary discussions with experimentalists confirm their interest and validation of the model and analysis.

Future work includes adding explicit derivatives, so that we can reason about the amplitude of the pulsation. We will refine the model with data on the rates, as the data becomes available, and add more detail such as it requires 4 molecules of cAMPs to activate one PKA. We will also investigate the value of adding second derivatives to identify local minima and maxima.

We have investigated further hypotheses, such as whether the way PDE8A1 decreases the PKA phosphorylation of Raf-1 at S259 can be counterbalanced by the addition of a PDE8A1 inhibitor such as the a drug Dipyridamole. This is the topic of a further paper.

Acknowledgements

We would like to thank Walter Kolch, George Baillie and Kim Brown from the Faculty of Biomedical & Life Science, University of Glasgow, for discussions, guidance and insight into the AKAP scaffold.

This research is supported by the *SIGNAL* project, funded by the Engineering and Science Research Council (EPSRC) under grant number EP/E031439/1.

References

[BHHK03] Christel Baier, Boudewijn R. Haverkort, Holger Hermanns, and Joost-Pieter Katoen. Model-Checking Algorithms for Continuous-Time Markov Chains. *IEEE Trans. Software Eng.*, 29(6):524–541, 2003.

[BMM09] Paolo Ballarini, Radu Mardare, and Ivan Mura. Analysing Biochemical Oscillation through Probabilistic Model Checking. *Electr. Notes Theor. Comput. Sci.*, 229(1):3–19, 2009.

[CDHC09] Federica Ciocchetta, Andrea Degasperi, Jane Hillston, and Muffy Calder. Some Investigations Concerning the CTMC and the ODE Model Derived From Bio-PEPA. *Electr. Notes Theor. Comput. Sci.*, 229(1):145–163, 2009.

[CRCFS04] Nathalie Chabrier-Rivier, Marc Chiaverini, Vincent Danos François Fages, and Vincent Schächter. Modeling and querying biomolecular interaction networks. *Theoretical Computer Science*, 325(1):25–44, 2004.

[CVGO06] Muffy Calder, Vladislav Vyshemirsky, David Gilbert, and Richard J. Orton. Analysis of Signalling Pathways Using Continuous Time Markov Chains. In Corrado Priami and Gordon D. Plotkin, editors, *T. Comp. Sys. Biology*, volume 4220 of *Lecture Notes in Computer Science*, pages 44–67. Springer, 2006.

[Fag05] François Fages. Temporal Logic Constraints in the Biochemical Abstract Machine BIOCHAM. In Patricia M. Hill, editor, *LOPSTR*, volume 3901 of *Lecture Notes in Computer Science*, pages 1–5. Springer, 2005.

[JEF00] Jr. James E. Ferrell. What Do Scaffold Proteins Really Do? *Sci. STKE*, 2000(52):1–3, 2000.

[KNP07] Marta Z. Kwiatkowska, Gethin Norman, and David Parker. Stochastic Model Checking. In Marco Bernardo and Jane Hillston, editors, *SFM*, volume 4486 of *Lecture Notes in Computer Science*, pages 220–270. Springer, 2007.

[MRM+08] Pedro T. Monteiro, Delphine Ropers, Radu Mateescu, Ana T. Freitas, and Hidde de Jong. Temporal logic patterns for querying dynamic models of cellular interaction networks. *Bioinformatics*, 24(16):227–233, 2008.

[RBFS08] Aurélien Rizk, Grégory Batt, François Fages, and Sylvain Soliman. On a Continuous Degree of Satisfaction of Temporal Logic Formulae with Applications to Systems Biology. In Monika Heiner and Adelinde M. Uhrmacher, editors, *CMSB*, volume 5307 of *Lecture Notes in Computer Science*, pages 251–268. Springer, 2008.

Towards a process-calculi approach to study the evolution of biological networks

Alessandro Romanel

CoSBi and Universitá di Trento, Italy

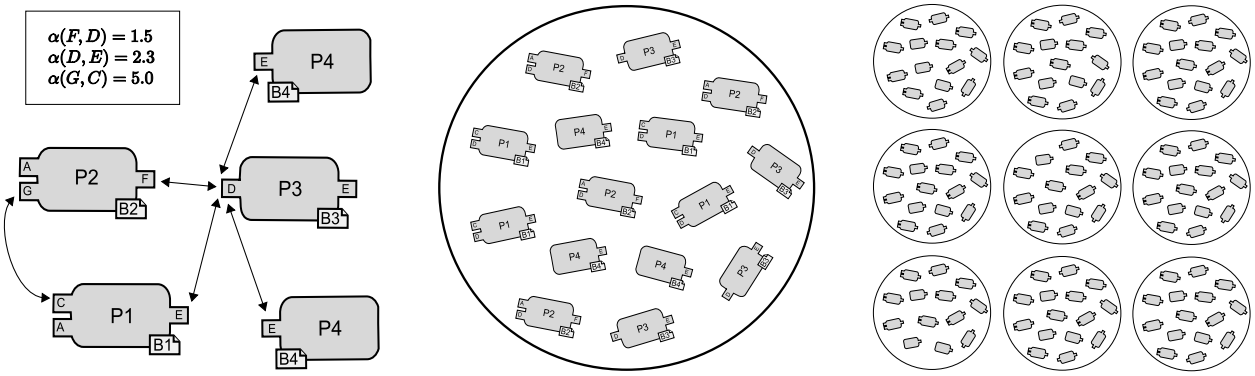
Over the last decade an increasing interest in using evolutionary approaches to study biological networks has continuously grown. Understanding how networks emerged during evolution can help us to understand their basic properties, such as the role of complexity and the importance of topology and feedback loops.

In [5] we developed a specific framework to allow straightforward study of network evolution based on BlenX [4] and the Beta Workbench [3], a process calculi based programming language and its stochastic simulation engine, respectively. We proposed a framework for simulating the evolution of protein-protein interaction networks where evolution proceeds through selection acting on the variance generated by random mutation events, and individuals replicate in proportion to their performance, referred to as fitness. This follows the idea that in order to simulate evolution by natural selection, we must be able to express populations of individuals, variability and fitness.

BlenX represents a protein as a computational entity, a *box*, composed by a set of *interfaces* and an *internal program*. Interfaces, which represent protein domains, have associated a *sort* (i.e. representing the structure of the domain) and are the places where a protein interacts with other proteins; the internal program, instead, codifies for the mechanism that transforms an interaction into a protein conformational change, which can result in the modification of other interface sorts.

A BlenX program specifies qualitatively and quantitatively a *system* of proteins and their interaction capabilities. In particular, the former are specified through a binary relation α on interface sorts (see [4] for details). The graphical notation of boxes and their interaction capabilities we use throughout this paper is depicted in Fig.1(a). Since in our evolutionary framework a system specified by a BlenX program represent and individual (see Fig.1(b)), a population consists of a set of different BlenX programs, each representing an individual composing the population (see Fig.1(c)).

In [5] the evolution of a population is implemented with an evolutionary algorithm which works in four main parts and is iterated for a specified number of steps; each iteration is called *generation*. The algorithm firstly generates the initial population; the population can be generated randomly, from a predefined network configuration to be used as a starting point, or it can be a network with no interactions. Each individual in the population is



(a) Boxes and boxes interactions (b) Individual (c) Population

Figure 1: a) The small squares on the border of boxes are the interfaces; `C`, `D` `E`,... are the interface structures (omitted when not necessary); `P1`, `P2`,... are internal programs and `B1`, `B2`,... are the names of the boxes (protein species). The arrows are the graphical representation of the compatibility relation α (on top) which describes qualitatively and quantitatively boxes interaction capabilities, i.e., through which interface pairs two boxes can exchange signals and communicate. b-c) Examples of an individual and a population of different individuals.

then simulated separately using the Beta Workbench stochastic simulator, and the outputs of the simulations are used to compute the fitness values of the individuals. Like in a real environment, individuals with the highest fitness values are more likely to survive, replicate and produce a progeny that resembles them, being not, however, completely equal to them. This part of the algorithm, indeed, creates a new population with the same number of individuals of the current population, using as a base the current individuals. Depending on a probability proportional to the fitness measure, individuals replicate and pass to the next generation. During the replication, each protein in the individual is given the chance to mutate, according to a probability. The different types of mutations we considered in [5] are based on real biological processes where mutations can happen at *DNA* and protein level. Variability is achieved by associating each of the considered mutations to a BlenX program modification. The great flexibility of BlenX in the definition of the structure of proteins, indeed, allow us to introduce primitives for mutations used to build domain-based interaction and mutation models. Starting from the study of mutations at a biological level, we end up with some interesting program modifications that permit us to mutate the BlenX representation of proteins in a meaningful and automatic way.

It is clear that the measure of fitness is problem dependent: it varies with the kind of network, with the characteristics a scientist wants to investigate, and so on. This measure can be done in various ways, including stability analysis, integration of the signal, measure of the derivative. In [5] the fitness was computed using integration of protein species time-courses in stochastic simulation results. Here we propose and discuss a theoretical framework, based

on concurrency theory, for computing certain classes of fitness measures. The goal of this paper is to present intuitively and informally most of the ideas we want our framework to be based on.

The dynamics of a system implemented by a BlenX program is described by a stochastic structural operational semantics, which distinguishes between monomolecular reactions (a single box is involved), bimolecular reactions (two boxes interact by means of synchronization or communication) and events (global box rewriting rules), allowing the generation of transition systems with arcs labelled by stochastic rates. This dynamics results exactly in the *behaviour* we are interested to study, i.e., trajectories in transition systems encode the discrete variation of protein species and protein domains amounts. In particular, we are interested in determining if the dynamics of protein species and protein domains in a given system *fits* an ideal behaviour, i.e., a certain time-course representing the ideal performance (see Fig.2). When we say fits, we intrinsically admit an amount of error, hence opening the doors to the realm of approximation methods [1, 2, 8, 6]. These methods aim to bridge the gap between rigid equivalence checking techniques and more relaxed requirements of real systems, and some of them deal with notions of behavioral equivalence for deciding if two systems behave almost (up to small errors or fluctuations) the same or, more formally, for measuring the distance between systems. One well-established approach uses pseudometrics, which give a measure of the similarity of systems that are not equivalent (see e.g. [8, 6]).

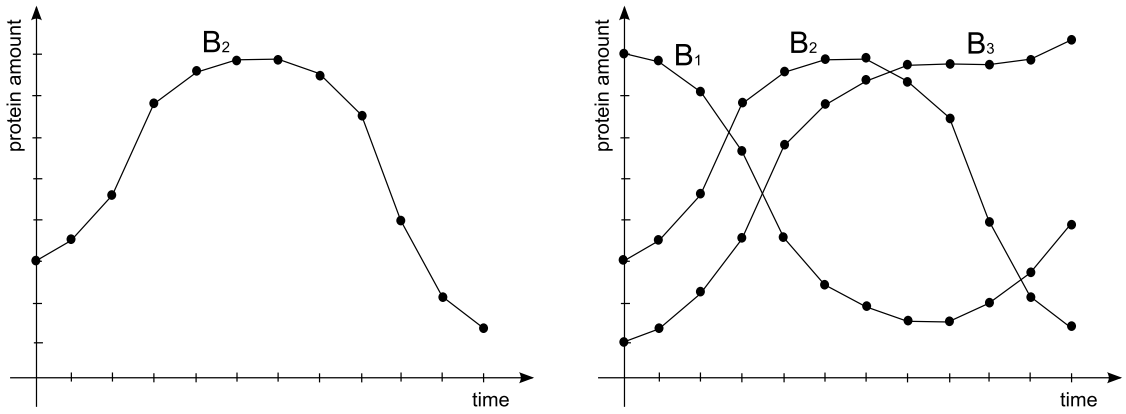


Figure 2: a) Example of ideal behaviour of a single protein species; b) Example of ideal behaviours of a set of protein species.

At this stage, anyway, we are not interested in comparing systems, but only systems against ideal behaviours. We rely on an approximated variant of the *simulation preorder* notion for deciding if a system *almost* simulates (fits) a certain ideal behaviour. Denoting with S_i a i-th discrete configuration of a system described in BlenX, an ideal behaviour (or more generally a set of ideal behaviours) can be represented as a *trace* of the form:

$$T = S_0 \xrightarrow{t_0} S_1 \xrightarrow{t_1} S_2 \xrightarrow{t_2} \cdots \xrightarrow{t_{n-1}} S_n$$

where t_i values represent times. It is clear how this trace definition captures the notion of time-course (or more generally of a set of time-courses). However, since we deal with times, it is quite obvious that traces can be used to represent behaviours at different time scales. For simplicity, we start by considering only traces at the same time scale of BlenX systems dynamics.

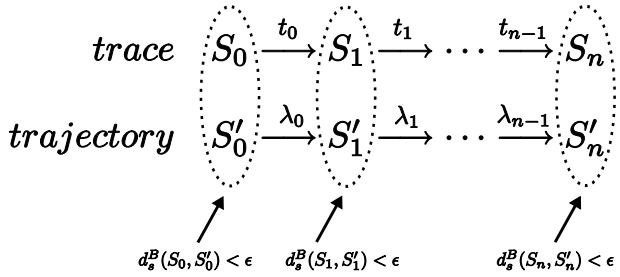
In order to develop further our fitness definition, we have first to introduce a notion of observables. At this stage we consider the two main properties we are interested to observe in a system configuration, namely the amount of protein species and protein domains. In our BlenX setting, classes of structurally equivalent boxes represent protein species while interfaces with same sorts represent protein domain classes. Denoting with S the system configuration reported in Fig.1(b), we indicate with $Obs_s(S, B_4)$ and $Obs_d(S, E)$ observations in S of, as an example, protein species B_4 and domains E, respectively, obtaining:

$$Obs_s(S, B_4) = 3 \quad \text{and} \quad Obs_d(S, E) = 11$$

Note that given a protein species B and a protein domain A, the functions:

$$d_d^B(X, Y) = |Obs_s(X, B) - Obs_s(Y, B)| \quad \text{and} \quad d_s^A(X, Y) = |Obs_d(X, A) - Obs_d(Y, A)|$$

are metrics on the set of BlenX system configurations. These notions of distance represent two of the ingredients we want to use to develop our approximated simulation relation. Given a bound ϵ, indeed, we can construct a relation that compares systems and ideal behaviours up-to fluctuations of protein species and protein domains amounts within the range ϵ. As an example, the following picture shows how a BlenX transition system trajectory match a trace up-to fluctuations in the amount of protein species B within the range ϵ.

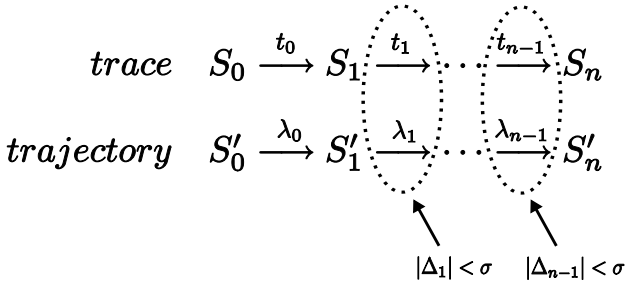


It is quite clear how traces differ from BlenX transition systems trajectories by the presence of times instead of stochastic rates on arc labels. Here we relate traces and trajectories by considering the expected values of the negative exponential distributions with parameters λ_i. Given a bound σ [1], if we have that the distances:

$$|\Delta_1 = t_1 - \frac{1}{\lambda_1}| \ , \quad |\Delta_2 = \Delta_1 + t_2 - \frac{1}{\lambda_2}| \ , \quad \cdots \ , \quad |\Delta_{n-1} = \Delta_{n-2} + t_{n-1} - \frac{1}{\lambda_{n-1}}|$$

are all less than σ, then we know that the trajectory has an average time behaviour which is close to the one of the trace up-to fluctuations within the range σ:

[1] Obviously, in formalizing these ideas we have to be sure that the size of σ respects some size relation with the traces times in order not to create inconsistencies in our calculations.



At this point, given a set of protein species and protein domains we are interested to observe, a BlenX system S' and a trace T, we can combine the two just given notions and say that *S' simulates trace T with precisions ϵ and σ, only if S' can match all the moves of T up-to fluctuations in observations and time values within ranges ϵ and σ, respectively*; Fig.3 gives graphical intuitions of what this relation looks like. By finding the tightest ϵ and σ such that S' simulates T we can build a notion of distance (metric) between BlenX systems and traces; this allows to compare different systems with respect to the distance they have from a trace, given the set of observations we are interested in. This concept fits perfectly in our evolutionary framework and can be rephrased in terms of comparing different individuals with respect to the distance they have from an ideal behaviour, i.e., the notion of approximated simulation can be used to compute the fitness of an individual. Tightest is the precision of the simulation, greater is the fitness value. Obviously, since the precision depends on two values, we can think of finding a formula that combines them (e.g. a linear combination) allowing for the weighting of the precisions importance.

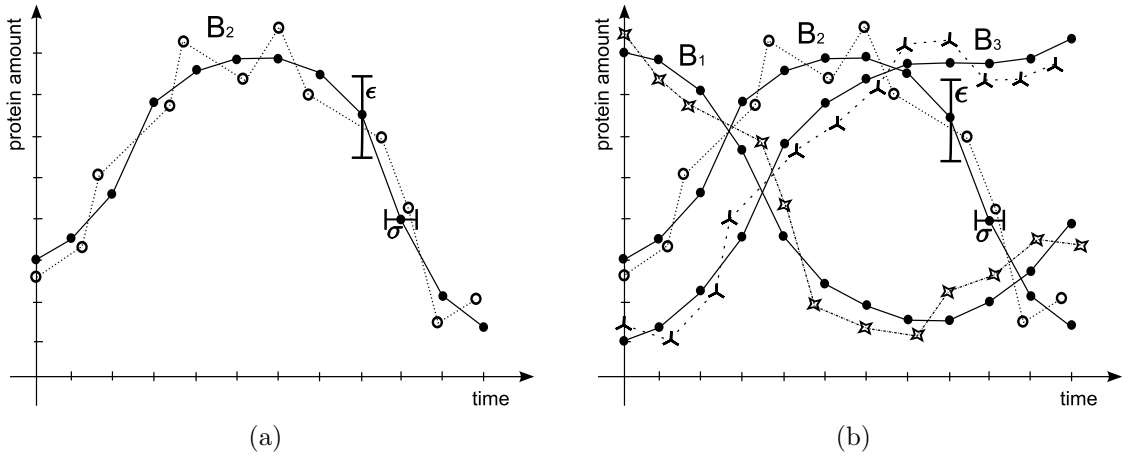


Figure 3: a) Example of simulation where only a protein species is observed; b) Example of simulation where a set of protein species is observed.

Note, anyway, that the given definition of simulation says only that in the transition system of S' there exists a trajectory that match the trace up-to fluctuations, but says anything about the probability to follow this trajectory with respect to all the possible trajectories

of length n in the transition system (n is the length of the trace). This is quite annoying, because in our fitness computation we would like to consider not only that a system almost simulates an ideal behaviour, but also that the simulation is not a rare event. In order to obtain this, the idea is to use the previous definition of simulation and compute the probability p that a system S' has to generate a trajectory that simulates T with given precisions ϵ and σ. Also in this case we can develop a notion of distance between systems and traces by combining not only precision values, but also the probability p; the idea is to find the tightest ϵ and σ and at the same time maximize p. Like before, we want a combination that allows weighting the three parameters. Note that by giving a weight zero to the probability value we should be able to recover the previous approximated simulation definition.

Although useful as a starting point to develop a theoretical framework for fitness computation, we think that this approach represents also a first step towards the achievement of certain kind of description of *neutrality* [9] in terms of some process-algebraic definition of functional or behavioural simulation or equivalence meaningful in the biological domain. This will clearly be of help in applying concurrency theory to study evolutionary robustness and evolvability, concepts recently recognized as crucial to the understanding of evolution [9]. We want to finish by mentioning a different process-calculi based work presented in [7], which aims to combine a variant of π-calculus, the continues π-calculus, and model-checking techniques as an approach to study robustness and evolvability of biological networks.

References

[1] M. Backes. Quantifying probabilistic information flow in computational reactive systems. In LNCS, editor, *Eur. Symp. on Research in Computer Security*, volume 3679, page 336354, 2005.

[2] C. Baier, J.P. Katoen, and H. Hermanns. Approximate symbolic model checking of continuous time markov chains. In LNCS, editor, *Conf. on Concurrency Theory*, volume 1664, page 146162, 1999.

[3] L. Dematté, C. Priami, and A. Romanel. The Beta Workbench: a computational tool to study the dynamics of biological systems. *Brief. Bioinform.*, 9(5):437–449, 2008.

[4] L. Dematté, C. Priami, and A. Romanel. The BlenX Language: A Tutorial. In LNCS, editor, *SFM 2008*, pages 313–365. Springer-Verlag, 2008.

[5] L. Dematté, C. Priami, A. Romanel, and O. Soyer. Evolving BlenX programs to simulate the evolution of biological networks. *Theor. Comput. Sci.*, 408(1):83–96, 2008.

[6] J. Desharnais, V. Gupta, R. Jagadeesan, and P. Panangaden. The metric analogue of weak bisimulation for probabilistic processes. In *Symp. on Logic in Computer Science*, volume 413422, 2002.

[7] M. Kwiatkowski and I. Stark. The continuous pi-calculus: A process algebra for biochemical modelling. In *CMSB*, pages 103–122, 2008.

[8] F. van Breugel, C. Hermida, M. Makkai, and J. Worrell. An accessible approach to behavioural pseudometrics. In LNCS, editor, *Colloquium on Automata, Languages, and Programming*, volume 3580, 2005.

[9] A. Wagner. *Robustness and Evolvability in Living Systems (Princeton Studies in Complexity)*. Princeton University Press, August 2005.

Spatial extension of stochastic π calculus

Anton Stefanek Maria G. Vigliotti Jeremy T. Bradley

Department of Computing, Imperial College London

{as1005|mgv98|jb}@doc.ic.ac.uk

August 18, 2009

Abstract

We introduce a spatial extension of stochastic π-calculus that provides a formalism to model systems of discrete, connected locations. We define the extended stochastic semantics and also give deterministic semantics in terms of a system of ordinary differential equations. We describe two simple examples, one based on a standard epidemic model and one modelling resistance in plant tissues.

1 Introduction

Stochastic process algebras are becoming increasingly important in Systems Biology. Several frameworks have been developed (such as Bio-PEPA [6]) that allow convenient description of the models and different ways of formal analysis. Moreover, SPAs are naturally suited for extensions – for example in [5] Bio-PEPA is extended with spatial descriptions, a very important feature for modelling of various biological systems.

Stochastic π-calculus is a SPA that has been successfully applied to modelling in Systems Biology [4, 3, 16, 10]. This gives a motivation for its further extensions. We introduce certain spatial features to stochastic π-calculus. This has been done to some extent in the Bio-Ambient calculus [15]. However, our aim is to provide an extension that would allow a definition of an alternative, deterministic and continuous, semantics (it is not clear to us how this could be done for the Bio-Ambient calculus). In this work we will first remind the reader of the stochastic π-calculus and provide a definition of the deterministic semantics, based on the continuous π-calculus [11] and PEPA [9]. Then we introduce a spatial extension $\mathcal{L}\pi$ that allows modelling of discrete, connected locations and show how it keeps both the stochastic and deterministic semantics. We illustrate our ideas on two examples, one a standard epidemic model and another from plant physiology.

2 Stochastic π calculus

We will use a formalism (shortened to $\mathcal{S}\pi$) based on the standard stochastic π-calculus, as described in [14].

The basic primitives of $\mathcal{S}\pi$ are *processes* that communicate over *channels* or evolve independently. Communication happens via *actions*. On a channel a, a process can perform an *output action* $!a$, possibly involving transmission of a message, $!a\langle\widetilde{\psi}\rangle$ (where $\widetilde{\psi}$ is a vector of variables and channel names). On the other hand, a process can perform an *input action* $?a$, possibly receiving a message, $?a(\widetilde{x})$ (where $\widetilde{x}$ is a vector of variables which become bound by the values in the received message). Each channel has an associated constant rate, which corresponds to the rate of communication over that channel. To evolve independently, a process can perform a *silent action* $\tau@r$ at a specified rate r.

To summarize, the actions are

$$\alpha = !a \setminus ?a \setminus !a\langle\widetilde{\psi}\rangle \setminus ?a(\widetilde{x}) \setminus \tau@r.$$

The processes are built inductively from actions and the basic *zero* process $\mathbf{0}$ not capable of any action. A *continuation* is a process of the form $\alpha.P$ (where P is a process), capable of performing an action α and thereby evolving into the process P. A *summation* $\sum_{i\in I}\alpha_i.P_i$ is a

process with multiple such capabilities. A process $P|Q$ is a *parallel composition* of two processes P, Q that can communicate together. The restriction operator 'new' ensures that the channels in the set φ are private to the *restriction* process $(\text{new}\,\varphi)P$, unless sent to another process via channel communication. Finally, for a more convenient modelling and, more importantly, to allow recursion, a *process identifier instance* can be used in place of a process it defines, possibly with some parameters, $A\langle\widetilde{\psi}\rangle$.

To summarize, the set of processes of $\mathcal{S}\pi$ (denoted by $\mathcal{P}$) contains

$$P, Q = \mathbf{0} \setminus \sum_{i \in I} \alpha_i.P_i \setminus P|Q \setminus (\text{new}\,\varphi)P \setminus A\langle\widetilde{\psi}\rangle.$$

The binding between identifiers and the processes they define is specified in an *environment*, a collection of defining equations of the form

$$A(\widetilde{\psi}) \stackrel{\text{def}}{=} P.$$

An environment E together with an initial process S form a *system of* $\mathcal{S}\pi$ – a complete description of the underlying model.

For an example, consider a simple epidemic model. Due to a disease, a population can be divided into three types of individuals – *susceptible*, *infected* and *recovered*. A susceptible individual can catch the disease when meeting an infected one, who can also recover from the disease after a period of time. We can take the environment E_{SIR} consisting of the equations

$$\begin{aligned} S &\stackrel{\text{def}}{=} ?i.I, \\ I &\stackrel{\text{def}}{=} !i.I + \tau@r_{rec}.R, \\ R &\stackrel{\text{def}}{=} \mathbf{0} \end{aligned}$$

representing the three different types of individuals and specifying their behaviour. The communication on the channel i between the processes I and S corresponds to the transmission of the disease from an infected to a susceptible individual and the silent transition from a process I to the process R corresponds to the recovery of an infected individual. The population can be represented by an initial process of the form $s \times S|i \times I|r \times R$ (where $n \times P$ is a shorthand for a parallel composition of p copies of P).

2.1 Stochastic semantics

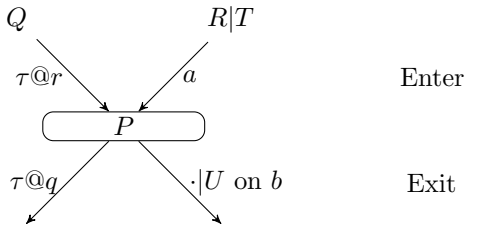
Traditionally, stochastic π-calculus has a discrete, stochastic semantics [14], given in the form of a continuous time Markov chain (CTMC). The states of the CTMC correspond to the processes and the transition rates are obtained from the rates associated to the channels and the rates of the silent actions according to rules defined on the structure of processes. Parallel compositions are capable of the same actions as their components. Those can can evolve independently or communicate on channels. If two processes capable of complementary actions (input and output) on the same channel are in a parallel composition, they evolve together with rate corresponding to the rate of the channel. If the process performing output action also sends a channel name coming from a restriction, the other process is put under the same restriction. Identifier instances are treated as the processes they define.

Unlike for some of the other SPAs, such as PEPA, the CTMCs arising from $\mathcal{S}\pi$ models can have an infinite state space (products of continuations can be parallel compositions) and so the only standard method of analysis is the stochastic simulation using a variant of the Gillespie algorithm [8]. The efficiency of this algorithm can be improved by aggregating identical processes within parallel compositions (using properties of the structural congruence for $\mathcal{S}\pi$) and hence making the complexity independent of individual process populations. The aggregation also shows that $\mathcal{S}\pi$ naturally obeys the law of mass action - it can be shown that the rate of communication between two processes depends on the product of their populations. See [17] for details.

2.2 Deterministic semantics

A recent trend in SPAs is to provide an alternative continuous semantics that serves as a deterministic approximation to the underlying CTMC, [9, 1, 11]. This has been done for an extension of the stochastic π-calculus, the *continuous π-calculus*, in [11]. We will employ a style more similar to the case of PEPA [9].

In the deterministic semantics, populations of processes are approximated by real valued functions over time that are mutually related via a system of ordinary differential equations (ODEs). For $\mathcal{S}\pi$, similarly to [11], we can define a notion of a *prime process* – a process that cannot be split into a parallel composition of non-zero processes – and show that each process (corresponding to a state in the CTMC) can be uniquely expressed as (is structurally congruent to) a parallel composition of prime processes. Then we can define for each prime process P a real valued function $[P]$ giving the population (its approximation) of P over time. To obtain the system of ODEs, we look at the possible ways the population of P can increase and decrease in a short period of time. It can increase as a result of a communication between two prime processes R, T or as a result of a silent transition of some prime process Q. On the other hand, it can decrease due to communication between P and another prime process Q or a silent transition of P.



We can keep track of these possibilities together with their multiplicities (for example $\tau@r.(P|P)$ increases the population of P by two) in form of Enter and Exit multisets and define the *system of ODEs of a system* (S, E) as consisting of the following, for each reachable prime process P:

$$\frac{\mathrm{d}[P]}{\mathrm{d}t} = \sum_{(r,Q)\in \mathrm{Enter}_{\tau,S,E}(P)} r \cdot [Q](t) + \sum_{(a,R,T)\in \mathrm{Enter}_{ch,S,E}(P)} r_a \cdot [R](t)[T](t)$$
$$\sum_{q\in \mathrm{Exit}_{\tau,S,E}(P)} q \cdot [P](t) - \sum_{(b,U)\in \mathrm{Exit}_{ch,S,E}(P)} r_b \cdot [P](t)[U](t).$$

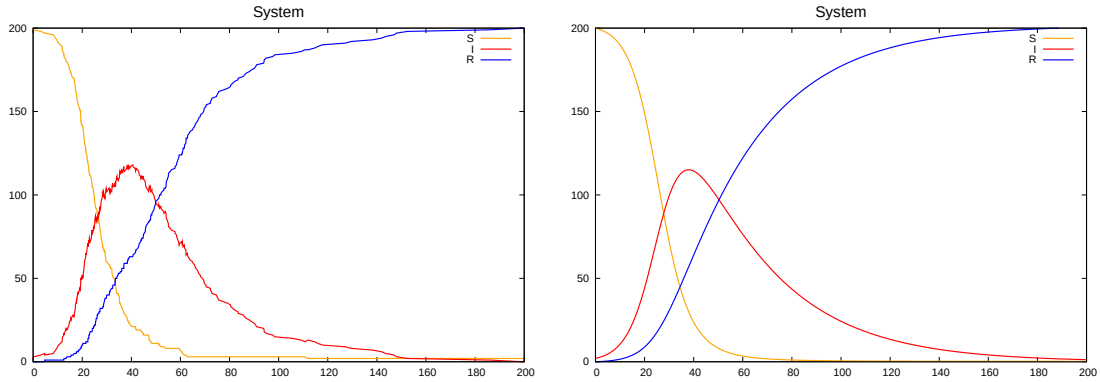
The initial values of the functions $[P]$ are given by the populations of P in the initial process S. This system of ODEs can then be numerically solved to give an approximation of process populations over time.

For the epidemic example, we get the following system of ODEs

$$\begin{aligned} \mathrm{d}[S]/\mathrm{d}t &= -r_i \cdot [S](t)[I](t), \\ \mathrm{d}[I]/\mathrm{d}t &= r_i \cdot [S](t)[I](t) - r_{rec} \cdot [I](t), \\ \mathrm{d}[R]/\mathrm{d}t &= r_{rec} \cdot [I](t). \end{aligned}$$

with initial values given by s, i and r in $s \times S|i \times I|r \times R$. See Figure 1 for a numerical solution of this system and a comparison with a sample stochastic simulation.

Unfortunately, the set of ODEs from the deterministic semantics of a $\mathcal{S}\pi$ system does not necessarily have to be finite. The interplay between the recursion and restriction can potentially result in an arbitrary number of newly created channels that are "connected" in arbitrary many different ways and thus form an arbitrary number of prime processes and the corresponding real valued functions. This is one of the distinctive features of stochastic π-calculus and has been used to model similar structures in nature, such as polymerization of actin filaments in [3]. However, most of the stochastic π-calculus models from the literature do not require such features (see [12] for a collection of some of them) – it makes sense to provide a characterization that will ensure a finite set of ODEs. A very crude solution is to restrict the stochastic π-calculus to the *Chemical*



(a) Stochastic simulation (b) ODE solution

Figure 1: A simple epidemic example. Comparison of the stochastic simulation of the CTMC from the discrete semantics and a numerical solution to the ODEs from the continuous semantics.

Ground Form (CGF) – a subset excluding the restriction operator and action parameters. We can show that this ensures finiteness of the resulting set of ODEs. This agrees with [2], where the author provides a translation of models in CGF to ODEs via chemical equations.

Finally, most of the models in [12] are in CGF or can be translated into "equivalent" ones in CGF (see [17] for some examples). This allows comparison of the two semantics and therefore further investigation into their relationship, with the possibility of formal limit results such as for PEPA [7].

3 Spatial extension – $\mathcal{L}\pi$

We can now define a simple spatial extension of $\mathcal{S}\pi$ which we will call $\mathcal{L}\pi$. We aim for a minimal extension that can provide a basis for further improvements. Considering applications in biology, we focus on *discrete* compartments between which the processes can move. The Bio-Ambient calculus [15] models this to a certain extent, but additionally allows dynamics of the compartmental structure. Albeit suitable for some applications (e.g. membrane modelling), we believe that it would not be straightforward to maintain the deterministic semantics of such extension – we therefore consider compartments with fixed structure. This can be justified by the fact that the current knowledge about biological systems is limited and the existing models mainly describe fixed compartments (as is argued in [5] where a similar extension is provided for Bio-PEPA).

3.1 Location graphs

We can argue that in case the compartments are non-overlapping (but possibly nested), their structure can be represented by a graph – the vertices represent the compartments and the edges represent how the processes (molecules, proteins, etc.) can move between them. This will form the basis of our extension. We define *location graphs* that give the structure of the *locations* (by which we mean generalized "compartments", not necessarily having physical boundaries). Inside each location, there are $\mathcal{S}\pi$ processes that can independently evolve, with rates affected by the locations volume. Moreover, these processes are allowed to move between the locations, with rate given by the location graph. We also assume that communication can happen only between processes inside the same location.

Formally, location graphs are of the form

$$[l_1 \colon P_1, \ldots, l_n \colon P_n]_{v,m}$$

where $L = \{l_1, \ldots, l_n\}$ is the set of location names, P_i are $\mathcal{S}\pi$ processes, $v \colon L \to \mathbb{R}$ is a *volume function* assigning a fixed volume to each location and $m \colon \mathcal{P} \times L \times L \to \mathbb{R}$ is a *movement function*

giving the rate of movement of processes between pairs of locations. A *system of* $\mathcal{L}\pi$ consists of an environment and a location graph.

Returning to the epidemic example, we can be interested in a system with a quarantine where the infected individuals get placed after a period of time and are kept until they recover. We can model this with a location graph with two locations, one corresponding to the original "world" (say a) and the other to the quarantine (say b), with processes coming from the original environment. There are two possible movements in this model – of I processes from a to b and of R processes from b to a. The rate of the first, say $r_{diagnose}$ can correspond to how fast an infected individual gets diagnosed and the rate of the second, say $r_{recover}$ to how long it takes to verify a recovery. Therefore we take $m(I, a, b) = r_{diagnose}$ and $m(R, b, a) = r_{recover}$ and $m(P, l_1, l_2) = 0$ for all the other processes P and locations l_1, l_2. We can ignore the location volume and set $v(a) = v(b) = 1$. As the initial location graph we can take (starting with an empty quarantine)

$$G = [a\colon s \times S | i \times I | r \times R, b\colon \mathbf{0}]_{v,m}$$

and get a system of $\mathcal{L}\pi$ (E_{SIR}, G).

3.2 Stochastic semantics

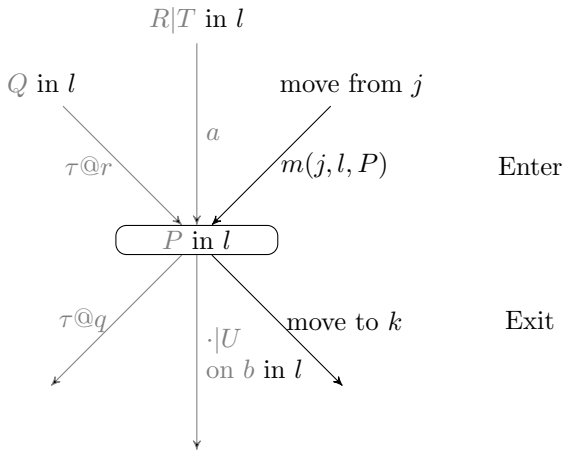
The stochastic semantics of $\mathcal{L}\pi$ is an extension of the stochastic semantics of $\mathcal{S}\pi$. Each state of the CTMC is a location graph. The transitions can be either internal to a single location or correspond to a movement between locations. In an internal transition, the processes inside a location l can communicate or evolve independently in the same way as in $\mathcal{S}\pi$, resulting in a transition to a location graph with only the contents of l changed accordingly. The rate is potentially affected by the volume of l – mimicking the chemistry, the rate of communication should be inversely proportional to the volume of l. Therefore the original $\mathcal{S}\pi$ rate gets divided by $v(l)$ in case it corresponds to a communication transition.

In the movement transition, a process P "moves" between two locations. If the process in l_1 contains P in a parallel composition and the movement function allows the movement of P between l_1 and another location l_2, i.e. $m(P, l_1, l_2) \neq 0$, then there is a possible transition to a graph with only locations l_1 and l_2 changed, where P is removed from the parallel composition in l_1 and added to the parallel composition in l_2. Ideally, we would only allow the movement of prime processes. However, the movement of restrictions would require more complicated semantics in order to respect the structural congruence. Therefore only summations and identifier instances defining summations can have a non-zero movement function. This can be further extended if suitable models that need movement of restrictions (complexes) are described.

The aggregation results and hence the efficient Gillespie algorithm from $\mathcal{S}\pi$ can be re-formulated for $\mathcal{L}\pi$ in an obvious way.

3.3 Deterministic semantics

The deterministic semantics of $\mathcal{L}\pi$ is an obvious extension. By the properties of $\mathcal{S}\pi$, the process in each location can be uniquely expressed as a parallel composition of prime processes. Therefore we can take a real valued function $[P]_l$ for each process P and location l. Population of a process P in each location l can change due to an internal transition – the corresponding terms in the resulting ODE are the same – and due to movement of P to and from l.



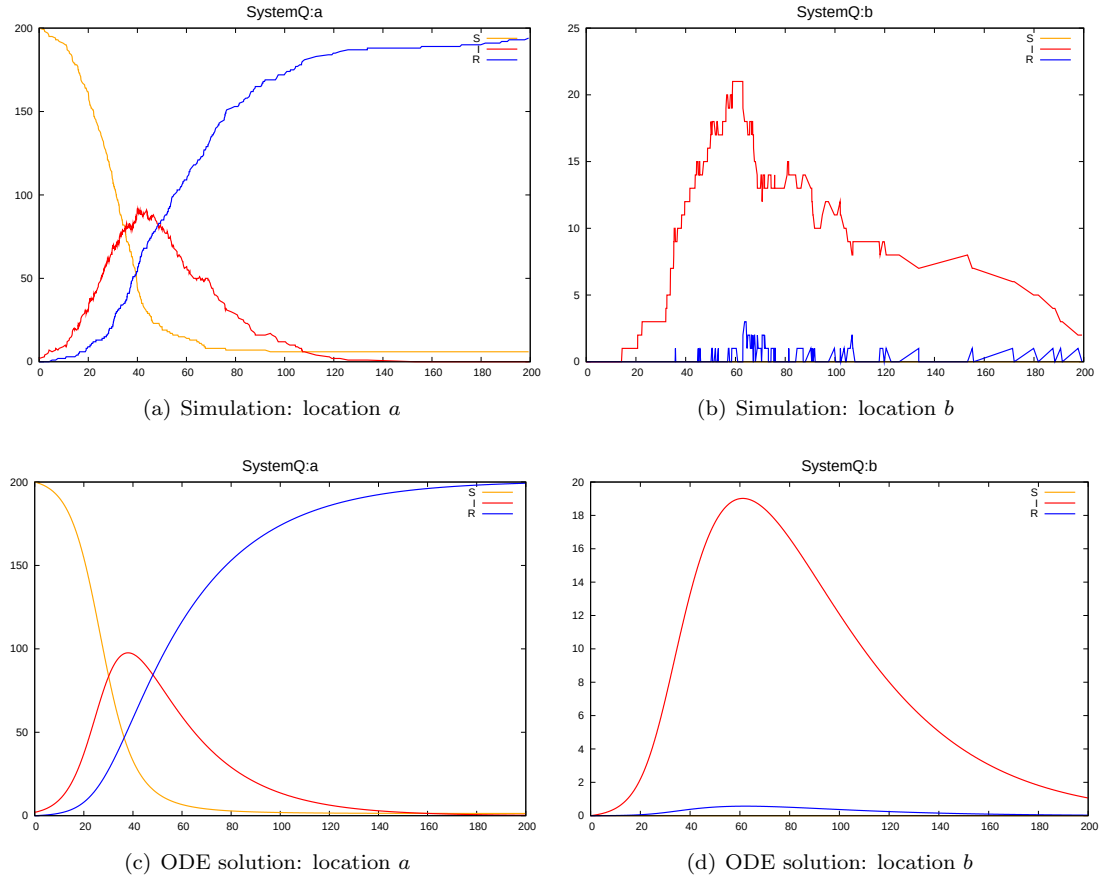
The Enter and Exit multisets can be constructed in a similar way to $\mathcal{S}\pi$, with the difference that we need to consider the movement in the set of all reachable prime processes. This then leads to an ODE for each real valued function $[P]_l$:

$$\begin{aligned}\frac{\mathrm{d}[P]_l}{\mathrm{d}t} = &\sum_{(r,Q)\in \mathrm{Enter}_{\tau,S,E}(P)} r[Q]_l(t) + \sum_{(a,R,T)\in \mathrm{Enter}_{ch,S,E}(P)} r_a[R]_l(t)[T]_l(t)/v(l) \\ &- \sum_{q\in \mathrm{Exit}_{\tau,S,E}(P)} q[P]_l(t) - \sum_{(a,U)\in \mathrm{Exit}_{ch,S,E}(P)} r_b[P]_l(t)[U]_l(t)/v(l) \\ &+ \sum_{m(j,l,P)\neq 0} m(j,l,P)[P]_j(t) - \sum_{m(l,k,P)\neq 0} m(l,k,P)[P]_l(t)\end{aligned}$$

For the epidemic example we get real valued functions $[S]_a, [I]_a, [R]_a, [I]_b, [R]_b$ (we ignore $[S]_b$ as, due to the definition of m, it clearly stays constant zero all the time). See Figure 2 for a numerical solution to the extended set of ODEs and a comparison with a sample simulation.

We will look at a simplified example from plant physiology [18]. Consider a hypothetical plant tissue consisting of cells arranged in a two dimensional grid. A cell can be attacked by a virus. A hypothesis is that in this case, it sends out a signal to the neighbouring cells, which in turn become more resistant to the virus and thus eventually prevent its spreading to the whole tissue. We will show how $\mathcal{L}\pi$ can be used to model this situation and so to carry out experiments in-silico to confirm the hypothesis.

Our location graph will represent the structure of the tissue – we take a grid with only adjacent nodes connected. Each location will correspond to a cell – initially, it will contain a *Cell* process. The *Virus* process will be able to attack a cell when in the same location. In that case, the cell releases warnings to the neighbouring cells – it will create several *Warning* processes, that are allowed to move to the neighbouring locations – and starts fighting the virus. The life of the cell will be represented by the process *Life* and the resistance against the virus by the *Resistance* processes. The resistance processes will be able to attack the virus (output action on the channel *defeat*), while the virus attacks the life of the cell (output action on the channel *fight*) – the likelihood of the cell surviving therefore depends on the number of resistance processes it releases. When the virus wins, the cell gets defeated and the virus multiplies, otherwise a resistance process destroys the virus and notifies the cell (output action on the channel *defeated*) which then switches back to the normal state (but with resistance to the virus). When a cell gets warned (communicates with a warning process), it switches to the resistant state (the process *RCell*), which is identical to the *Cell* with the difference that it releases more *Resistance* processes.



(a) Simulation: location a (b) Simulation: location b

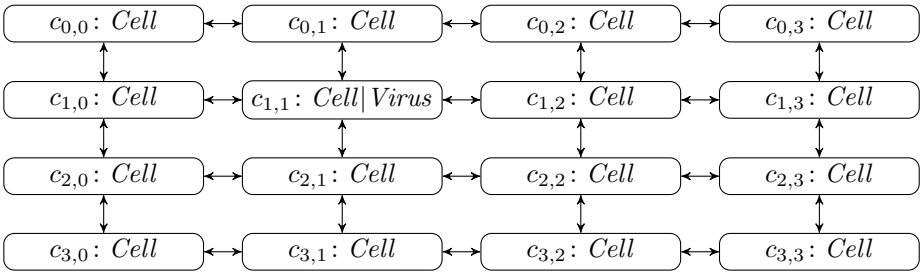
(c) ODE solution: location a (d) ODE solution: location b

Figure 2:

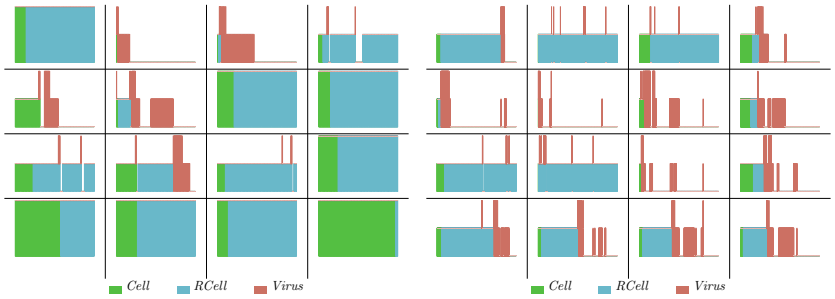
To model this behaviour we take an environment consisting of the defining equations

$$\begin{aligned} Cell &= ?attack.(Life|6 \times Reistance|4 \times Warning) + ?warn.RCell, \\ RCell &= ?attack.(Life|20 \times Reistance|4 \times Warning) + ?warn.RCell, \\ Resistance &= !defeat.!defeated + delay@expire, \\ Life &= ?fight + ?defeated.RCell, \\ Virus &= !attack.(!fight.(2 \times Virus) + ?defeat). \end{aligned}$$

To model the tissue structure, we take a location graph which can be drawn as the following:



The volume of all locations is a constant 1 and the *Warning* and *Virus* processes are allowed to move on the edges. See Figure 3 for a sample simulation of this $\mathcal{L}\pi$ system.



(a) Virus contained (b) Virus spreading

Figure 3: Sample simulation of the plant patogen model. Each cell in the grid represents time evolution of the corresponding compartment. The system starts with the virus in the location $c_{1,1}$. Figure (a) shows an example of a simulation where the virus is contained after attacking the neighbouring cells of $c_{1,1}$. Figure (b) shows a simulation where the virus spreads to the neighbouring cells and survives.

4 Conclusion and Further work

We introduced $\mathcal{L}\pi$, an extension of stochastic π-calculus that provides basic features for modelling systems of discrete, connected locations. To support this formalism, we developed a tool *JSPiM* (to be released in due time, see [17] for details of the implementation) that allows simulation and ODE generation and numerical solution of $\mathcal{S}\pi$ and $\mathcal{L}\pi$ models. Written in Java, it also serves as an alternative to the existing stochastic π-calculus tool *SPiM* [13]. We hope that with the help of this tool, more realistic examples from biology can be proposed, thus verifying the suitability of $\mathcal{L}\pi$ and giving direction for possible extensions. These could include the already mentioned movement of restrictions (usually representing chemical complexes). A syntactical extension of $\mathcal{L}\pi$ providing constructs for active movement could also prove useful – for example in the plant tissue model, the warning cells released would be each directed to a different neighbouring location instead of just allowed to randomly move.

A more ambitious task would be to relate $\mathcal{L}\pi$ to reaction-diffusion systems and work towards a formalism that would model the space continuously.

References

[1] L. Bortolussi and A. Policriti. Stochastic concurrent constraint programming and differential equations. *Electr. Notes Theor. Comput. Sci.*, 190(3):27–42, 2007.

[2] L. Cardelli. From processes to odes by chemistry. In *IFIP TCS*, 2008.

[3] L. Cardelli, E. Caron, P. Gardner, O. Kahramanoğulları, and A. Phillips. A process model of actin polymerisation. In N. Cannata, E. Merelli, and I. Ulidowski, editors, *Proceedings of the Workshop "From Biology To Concurrency and back (FBTC 2008)", July 2008*, volume 229 of *Electronic Notes in Theoretical Computer Science*, pages 127–144, Reykjavik, Iceland, 2008. Elsevier.

[4] L. Cardelli, P. Gardner, and O. Kahramanoğulları. A process model of rho gtp-binding proteins in the context of phagocytosis. *Electron. Notes Theor. Comput. Sci.*, 194(3):87–102, 2008.

[5] F. Ciocchetta and M. L. Guerriero. Modelling biological compartments in bio-pepa. *Electron. Notes Theor. Comput. Sci.*, 227:77–95, 2009.

[6] F. Ciocchetta and J. Hillston. Bio-pepa: a framework for the modelling and analysis of biological systems, 2008. Theoretical Computer Science.

[7] N. Geisweiller, J. Hillston, and M. Stenico. Relating continuous and discrete PEPA models of signalling pathways. *Theor. Comput. Sci.*, 404(1-2):97–111, 2008.

[8] D. Gillespie. Exact stochastic simulation of coupled chemical reactions. *The Journal of Physical Chemistry*, 81(25):2340–2361, 1977.

[9] J. Hillston. Fluid flow approximation of PEPA models. In *Proceedings of the Second International Conference on the Quantitative Evaluation of Systems*, pages 33–43, Torino, Italy, Sept. 2005. IEEE Computer Society Press.

[10] C. Kuttler and J. Niehren. Gene regulation in the pi calculus: simulating cooperativity at the lambda switch. *Transactions on Computational Systems Biology VII*, 4230:24–55, 2006.

[11] M. Kwiatkowski and I. Stark. The continuous π-calculus: A process algebra for biochemical modelling. In *Computational Methods in Systems Biology: Process of the Sixth International Conference CMSB 2008*, number 5307 in Lecture Notes in Computer Science, pages 103–122. Springer-Verlag, 2008.

[12] A. Phillips. Examples in spim. `http://research.microsoft.com/en-us/projects/spim/examples.pdf`.

[13] A. Phillips and L. Cardelli. Efficient, correct simulation of biological processes in stochastic π-calculus. *Proceedings of Computational Methods in Systems Biology*, pages 184–199, 2007.

[14] C. Priami. Stochastic π-calculus. *The Computer Journal*, 38(7), 1995.

[15] A. Regev, E. Panina, W. Silverman, L. Cardelli, and E. Shapiro. Bioambients: An abstraction for biological compartments. *Theoretical Computer Science, Special Issue on Computational Methods in Systems Biology*, 325(1):141–167, 2004.

[16] N. Segata and E. Blanzieri. Stochastic pi-calculus modelling of multisite phosphorylation based signalling: The pho pathway in sccharomyces cerevisiae. *Lecture Notes in Computer Science*, 2008.

[17] A. Stefanek. Continuous and spatial extension of stochastic π-calculus. Master's thesis, Imperial College of Science, Technology and Medicine, 2009.

[18] L. Taiz and E. Zeiger. *Plant Physiology*. Sinauer Associates, 2006.

How restrictive is the current action decomposition property for compression bisimulation?

Vashti Galpin
LFCS, School of Informatics
University of Edinburgh
Vashti.Galpin@ed.ac.uk

Bio-PEPA is a stochastic process algebra for modelling biological systems. Semantic equivalences such as bisimulation are defined for process algebras and allow for the comparison of different models with respect to their behaviour. A question of interest is what semantic equivalences are useful in systems biology modelling. There are three approaches that can be taken in answering this question. First, one can consider the existing equivalences used in computer science and second, one can develop equivalences based on what biologists view as equivalent behaviour. The third approach involves considering two systems that we expect to have the same behaviour and use that as the basis for an equivalence. This approach has been successfully applied to Bio-PEPA to define compression bisimulation.

Bio-PEPA avoids the state space explosion in two ways: by allowing analysis based on ordinary differential equations and by discretising the concentrations of species. In the second case, we can use different step sizes to obtain different discretisations of the same Bio-PEPA model. Even though these models have different discretisations, we expect them to have the same behaviour and it is possible to construct an equivalence that identifies two different discretisations of the same model. This equivalence is called compression bisimulation (Galpin and Hillston, Discretisation and equivalence in Bio-PEPA, to appear in Proceedings of CMSB 2009) and is qualitative in that it only considers reaction names and not reaction rates.

Compression bisimulation involves equivalence classes of states in the labelled transition system obtained from the Bio-PEPA model. States are considered equivalent if they have the same reactions available. Transitions are then generated between classes of states from the transitions of the labelled transition system, creating a new transition system. Classical bisimulation as defined by Milner is then used to determine if the new transitions systems obtained for each discretisation have the same behaviour.

It is possible to show that two discretisations of a single species are compression bisimilar without any restrictive conditions. However, when showing that compression bisimulation is a congruence with respect to the synchronisation operator, a condition called the current action decomposition property (CADP) is currently required. This property requires that if we have two Bio-PEPA models $P_1 \underset{L}{\bowtie} Q_1$ and $P_2 \underset{L}{\bowtie} Q_2$ that are both derived from the same Bio-PEPA model $P \underset{L}{\bowtie} Q$ and which both have the same reactions available, that P_1 and P_2 have the same reactions available and Q_1 and Q_2 have the same reactions available.

The investigations of this property have shown so far that for a reaction $\alpha \notin L$ there are two basic cases where CADP does not hold and for $\alpha \in L$ there are two basic cases where CADP does not hold. The first two cases are easy to exclude from consideration since they are not reasonable Bio-PEPA models. In these cases, we can show that α appears in P and in Q but we know that $\alpha \notin L$. For $P \underset{L}{\bowtie} Q$ to be a reasonable representation of a biological system, any shared reaction should be in L. For the cases with $\alpha \in L$, there are a number of avenues to consider including the role of creation and degradation reactions, and whether a Bio-PEPA model is fully expressible. These points will be elucidated during the presentation. This is work in progress.

An Overview of the Bio-PEPA Eclipse Plug-in

Adam Duguid

The School of Informatics, Laboratory of Computer Science, The University of Edinburgh a.j.duguid@sms.ed.ac.uk

Abstract. Tool support for any modelling language is vital if the language is to be used. Bio-PEPA is a relatively new language designed for the modelling of biological systems, with tool support for the Bio-PEPA language under development. Here we present the syntax allowed by the current version of the Bio-PEPA Eclipse Plug-in and give a brief overview of the features present within the plugin. The purpose of the paper is to serve as the initial documentation until a more complete manual is written.

1 Introduction

Bio-PEPA [1] is a process algebra designed for the modelling of biological systems and has been used to model a wide range of systems including circadian clocks [2], signalling pathways [3, 4] and genetic networks [5]. The modelling of these systems was made possible through the development of tools for the Bio-PEPA language.

Currently there exist two main tools for working with Bio-PEPA models, the Bio-PEPA Workbench and the Bio-PEPA Eclipse Plug-in. While both are under active development, they each serve a particular role. The Bio-PEPA Workbench is a prototype tool for introducing new language features and types of analysis; the Bio-PEPA Eclipse Plug-in is an environment intended to target end-users wishing to model in Bio-PEPA. The recent release of version 0.1.0 of the plug-in includes several key improvements, including support for the Bio-PEPA locations extension [6] and an improved syntax. With these changes it becomes important to document the new syntax if the tool is to be used. The paper will cover the Bio-PEPA language followed by a brief overview of the plug-in.

2 The Bio-PEPA Language

A biological system can be encoded as a Bio-PEPA model by way of a 6-tuple $\langle \mathcal{V}, \mathcal{N}, \mathcal{K}, \mathcal{F}_R, Comp, P \rangle$, where:

- $\mathcal{V}$ is the set of locations
- $\mathcal{N}$ is the set of species attributes
- $\mathcal{K}$ is the set of parameter definitions
- $\mathcal{F}_R$ is the set of functional rates

– *Comp* is the set of species components
– P is the model component

Each of these sets results in a separate type of definition. For the description of the Bio-PEPA Eclipse Plug-in syntax, the following formatting will be applied: keywords will be typeset using `typewriter` and variable names and values in *italics*. Optional parts to the definition will be indicated by the use of angular brackets (⟨ ⟩). Where applicable, lists of possible tokens will use the curly brace ({ }) or in the case for more complex expressions be separated by a vertical line (|).

Legal identifiers (IDs) in Bio-PEPA follow the requirements for naming of variables in Java. The first character must be either an alphabetical character or an underscore (_) and subsequent characters may be alpha-numerics or the underscore character. In models with multiple compartments the naming convention for species is slightly more complicated, with different representations allowed in different places.

speciesID Referring to just the species identifier is seen as a global reference. If a species can exist in three locations and a reaction is stated in terms of the species then the reaction is assumed to be defined as occurring in all three locations independently of each other.

speciesID@locationID This allows a reference to a specific species in a specific location. If a reaction is defined in this manner it will not be enabled in other locations that the species can exist in.

***speciesID@locID*$_1$*,...,locID*$_n$ If a reference to a species in more than one location, but not all, is required then a comma delimited list can be used. This allows fine-grained control over where the definition in question is applicable.

From herein, where a species identifier is expected it shall be referred to as a *speciesID* and the legal forms will be explicitly stated.

To aid in the description of the different parts of a Bio-PEPA model, the following model shall be used as a running example. Fig. 1 shows a simple Bio-PEPA model of a Michaelis-Menten reaction. The model describes a single compartment where all the species reside along with three parameters, two required to describe a reaction using Michaelis-Menten kinetics and the third for the conversion of species P back into species S. The actual reactions are labelled as alpha and beta and the model also contains definitions for the three species required for the reactions to occur. The different parts of A Bio-PEPA model will now described in the order as seen in the 6-tuple.

Lastly, to avoid the repetition throughout the different definitions, it should be noted that all definitions, with the exception of the model component, are terminated by the use of the semi-colon (;). This particular convention follows on from PEPA.

2.1 Locations

Each location must encode an identifier, the parent location, the size of the location ($s \in \mathbf{R}$), the type the location represents (*kind*) and the step-size ($H \in$

```
// Locations
location main : size = 1, type = compartment;

// Parameters
v_M  = 1.0;
K_M = 1.0;
r_1   = v_M * 0.1;

// Functional rates
kineticLawOf alpha : fMM(v_M, K_M);
kineticLawOf beta  : r_1 * P@main;

// Species components
S = (alpha, 1) << S + beta >> S;
P = alpha >> P@main + beta << P;
E = alpha (+) E;

// Labelled composition definition
pathway ::= S@main[1000] <*> P@main[0];
// Model component
pathway <alpha> E@main[100]
```

Fig. 1. Bio-PEPA Michaelis-Menten example.

$\mathbf{N}^*$) for the location. As with all identifiers in a Bio-PEPA model, the identifier must be unique to the model. The parent identifier allows the generation of the location tree, where the mapping is one parent to many children and the parent should be seen to envelope the entire of the child or children. The type (or kind) of location can be any from the set of {membrane, compartment} and it is assumed that a membrane will act as the parent to a single compartment.

The concrete syntax, as accepted by the plug-in is as follows:

$$\texttt{location}\ \mathit{ID}\ \langle\texttt{in}\ \mathit{parentID}\rangle : \texttt{size} = \mathit{value}\ \langle,\ \texttt{step-size} = \mathit{value}\rangle \\ \langle,\ \texttt{type} = \{\texttt{membrane,compartment}\}\rangle;$$

Here we can see that the keyword `location` labels the purpose of this definition. The keyword must be proceeded by the ID for this new location. If the model defines multiple locations, the spacial location can be specified by adding the keyword `in` and the ID for the parent. By defining just the parent, the location tree can be generated by the parser. A colon is then used to separate the identifier (and the relative location if defined) from the properties of this definition, where multiple properties are comma separated. In the case of a location the size is the only required property (where $\mathit{value} \in \mathbf{R}$) while the type remains optional as it defaults to `compartment`. The `step-size` ($\mathit{value} \in \mathbf{N}^*$) is only required if performing analysis of a Bio-PEPA model with levels, with

more detail below. The definition is then terminated by the use of the semicolon as previously stated.

If no locations are defined the default location is used, which is equivalent to that seen in Fig. 1, a compartment labelled main with size equal to one (if required by the solver in question). The location definition in Fig. 1 could also have been written without the type property being set due to compartment being the default.

2.2 Species Attributes

The purpose of the species attributes definition is closely connected to the model component definition, as will be described later. The model component contains the information regarding population counts of the different species which, while sufficient for time-series analysis using others Ordinary Differential Equations (ODEs) or Stochastic Simulation Algorithms (SSAs) , does not allow analysis by Continuous Time Markov Chains (CTMCs) . For CTMC analysis the model must be bounded, requiring upper limits as to expected population counts. In connection with the step size of a specie's location this will allow the tool to calculate the number of levels to allow analysis by levels, the approach used in [4]. To keep the model component syntax as simple and clean as possible, this additional information is recorded in a species attribute definition. As the only supplementary information currently allowed pertains to CTMC analysis the entire definition is optional. Indeed, as of the current version of the Eclipse Plug-in (version 0.1.0) CTMC analysis is not possible so the definition is intended more for future use. Expected usage of the definition will include solving the CTMC, with packages such as the Matrix Toolkit for Java (MTJ) [7] or to allow exporting of the model into other tools such as PRISM [8]. Whilst the purpose of the current form is similar to that of the original definition [1], much of what was encoded has been removed due to redundancy in the information or because a more apt location has been utilised.

The concrete syntax, as accepted by the plug-in is as follows:

$$\texttt{species}\ \textit{speciesID} : \texttt{upper} = \textit{value}\langle,\ \texttt{lower} = \textit{value}\rangle;$$

As can be seen, if the definition is used, the only required attribute is the upper bound ($\textit{value} \in \mathbf{N}^*$) as the default lower bound is the value zero. If an improved lower bound is known then it can also be specified ($\textit{value} \in \mathbf{N}^*$). In regards to the species identifier, any of the different forms can be used. It should be noted though that if the definition is required i.e. for any form of CTMC analysis, then the bounds must be specified for every species in every location it exists in.

2.3 Parameter Definitions

Parameter definitions, being a staple part of every language, require little in the way of explanation and is not different here. The concrete syntax, as accepted

by the plug-in is as follows:

$$ID = expression;$$

where

$$\begin{aligned} expression = {} & int \mid float \mid ID \mid speciesID \mid (expr) \mid expr + expr \mid \\ & expr - expr \mid expr / expr \mid expr * expr \mid \ln(expr) \end{aligned}$$

As can be seen, an *expression* is any standard mathematical expression with the definition structured as it is because of the recursive nature of the expression (*expr* simply being an abbreviation for *expression*, and not a different set of allowable expressions). These expressions can reference other parameters, locations (size) and species (population counts) by their respective identifiers, where speciesID must refer to a species in a single location if multiple locations are specified. The ability to use functions is also present, with support growing as required. As of version 0.1.0 of the plug-in, support is limited to ln, while the next release will include others such as the floor and ceiling functions. Standard infix operators are present with the exception of the exponentiation (to be present in the next release). It should be made clear that any parameter that refers to a dynamic component e.g. a species is itself labelled dynamic for the purposes of compiling out static expressions. So in Fig. 1 the parameter r_1 will be evaluated at compile time to avoid needless recomputation during every step.

2.4 Functional Rates

One of the key differences between PEPA and Bio-PEPA is the use of functional rates and how they are handled. In PEPA the action name has no rate attached to it, merely acting as a label for identifying transitions. Instead the rate was part of the prefix operator, with the overall rate for any given action never explicitly defined. In Bio-PEPA the rate is explicitly defined within the model and attached to the name.

The concrete syntax, as accepted by the plug-in is as follows:

$$\texttt{kineticLawOf}\ ID : expression';$$

where expression extends the previous expression statement to also include predefined kinetic rates as shown here.

$$expression' = expression \mid \texttt{fMA}(expr) \mid \texttt{fMM}(expr, expr)$$

The functional rates, like locations, are clearly labelled through the use of the keyword `kineticLawOf` which is followed by its identifier. Following the syntax of the locations, the colon is used as the separator between the identifier and it's rate. As stated, legal expressions include all those allowed in the parameter definitions along with additional functions specific to commonly found

rates in the biological domain (mass-action and Michealis-Menten (`fMA`, and `fMM` respectively)) with the ability to add others (e.g. competitive inhibition or Hill kinetics) later.

Briefly covering the currently available rates:

Mass-action takes one parameter, r, with the overall rate for the reaction being the product of the rate and the population counts of all the reactants and modifier species involved in the reaction e.g. `fMA`(r), where the reaction involves species S_1 and S_2 interact to form S_3, would equate to a rate of $r \times S_1 \times S_2$ (assuming stoichiometric coefficients of one). If instead the reaction only involved S_1 as a reactant then the corresponding rate would be $r \times S_1$. In Fig. 1, reaction beta could have been also written as `fMA`(r_1).

Michaelis-Menten takes two parameters, v_M and K_M and requires a set number of species to perform specific roles within the reaction. One species is required to act as a reactant (S), another as the enzyme (E) and the last as the product (P) as seen in Fig. 1. The ordering of the parameters, along with the associated rate can be seen below.

$$\texttt{fMM}(v_M, K_M) = \frac{v_M \times S \times E}{K_M + S}$$

2.5 Species Components

The species component definition lists all the reactions a particular species is allowed to participate in and in what role. The list of actions are separated by the choice operator (+), behaving in an identical manner to the choice operator in PEPA.

The concrete syntax, as accepted by the plug-in is as follows:

$$ID \;=\; S; \qquad (\text{where } S \;=\; action \mid S + S)$$

Where an action is defined as

$$\begin{aligned} action = {} & (rateID, stoichiometry)\ op\ speciesID;\ \mid\ rateID\ op\ speciesID;\ \mid \\ & (rateID[locationID\ op'\ locationID],\ stoichiometry)\ (.)\ ID;\ \mid \\ & rateID[locationID\ op'\ locationID]\ (.)\ ID; \end{aligned}$$

with $op \in \{>>, <<, (+), (-), (.)\}$ and $op' \in \{- >, < -\}$.

The first *ID* refers to a species but in this particular definition is labelled as an *ID* to make it distinct from the the other species identifiers expected within a single species component definition. This identifier must take the first *speciesID* form of just the species name with no location. Thus each global species can have at most one definition, with the *speciesID* used to allow control over where a particular action is permissible. The initial identifier is then followed by the equals symbol and then a list of actions separated by the choice operator. with each action taking one of the forms shown above. In the case of non-transport definitions i.e. the first two in the action definition, the *speciesID* can take any of

the previously defined forms. Thus a single reaction can be defined as occurring in the global sense for the species, in a single location or in a subset of the locations where the species is present. The identifiers labelled *rateID* are normal identifiers, but ones that must refer to a previously defined functional rate and *stoichiometry* is the stoichiometric coefficient for the species in this reaction ($stoichiometry \in \mathbf{N}^*$).

With the operators, it is best to connect the textual representations back to the symbols used in the original Bio-PEPA definition.

Behaviour	Bio-PEPA symbol	ASCII representation
reactant	$\downarrow$	$<<$
product	$\uparrow$	$>>$
activator	$\oplus$	$(+)$
inhibitor	$\ominus$	$(-)$
modifier	$\odot$	$(.)$
unidirectional transportation	$\rightarrow$	$->$
bidirectional transportation	$\leftrightarrow$	$<->$

Using the example in Fig. 1, we can see the species P acts as a product in reaction *alpha*, but only within location main. This clearly has no impact in the simple example, being that there is only one defined location, but it does allow the syntax for species in particular locations to be shown. The species P also participates in reaction *beta*, this time acting as a reactant.

The example in Fig. 1 also highlights some of the abbreviations permissible in the definition. If a statement does not specify a stoichiometric coefficient the default value of one is used. The definition for species S shows the syntax for including the stoichiometric coefficient and the abbreviated form for when the stoichiometry is one, these being the first two forms shown in the action definition above.

Transportation is one of the current features not highlighted in the running example. Transportation represents the movement of a single species between two adjacent locations within the model, as described in [6]. In the defined syntax above (third and fourth action definitions) it requires two *locationID*s, the first acting as the source and the second as the target and must refer to locations that this particular species resides in. As the locations are embedded within the transportation action the identifier is simply the species name. The general modifier operator is used for transportation as while the levels of the species in the two locations change, the overall amount does not. Transportation can either be in a single direction or in both, with an example below in the concrete syntax.

$$\ldots + \text{gamma}[\text{main} -> \text{child}] \; (.) \; \text{E};$$

In this example, species E essentially has an additional reaction defined which is equivalent to the following:

$$\text{E@main} -> \text{E@child}$$

If the transportation was defined as being bidirectional then the reverse reaction would also be defined, with both occurring at rate gamma. In instances such as these it is advisable to take care with the rates; use of the mass-action rate kinetic would produce two rates, one where the population count of E@main is factored in and the other which would be dependent on E@child while a custom rate would be used as is in both directions.

2.6 Model Component

The model component is the final definition in a Bio-PEPA model, both in terms of the building blocks to be described here and in terms of required location within the model. Bio-PEPA imposes little ordering on the definitions with the exception that the model component definition is the final entry. At this point a distinction has to be made between the model component definition and labelled compositional definitions. To offer compositionality the tool needs to accept fragments of the model component and allow the assigning of a label for reference. Essentially the definitions are nearly identical, with the difference being the assignment on an identifier and the use of the termination symbol.

The concrete syntax for labelled compositional definitions, as accepted by the plug-in is as follows:

$$ID \ ::= \ P; \qquad (\text{where } P \ = \ ID \mid \ (P) \mid \ P < L > P \mid \ speciesID[value])$$

where

Behaviour	Bio-PEPA symbol	ASCII representation
cooperation	$\underset{\mathcal{L}}{\bowtie}$	$< L >$

Unlike the model component, labelled compositional definitions require an identifier for reference in the model component. The assignment symbol differs here from other definitions, taking the form of ::= which uniquely identifies the definition type. The compositional fragment comes next, taking one of the forms described above. It can consist of an identifier for another labelled compositional definition, single species or the synchronisation of several species or identifiers. For a single species, the identifier must refer to a specific species in a specific location (if locations are used). This limits the *speciesID* to one of the first two forms as described earlier. After the species identifier comes the population count, where $value \in \mathbf{N}_0$. Examples of this can be seen in Fig. 1, where the identifier pathway refers to a labelled compositional definition containing S@main[1000]

The cooperation operator represents the glue within the compositional definition, fulfilling an identical role to its counterpart in PEPA. Just as in PEPA, the cooperation set, indicated by the symbol L, can either be the wildcard token ($*$) or a comma delimited list of actions. In Fig. 1 we can see the use of the wildcard within the definition labelled pathway. This will cause the left and right trees to synchronise on any common action names. In the case of the example this would could have been expanded out to <alpha, beta>.

The model component follows the same syntax as the labelled compositional definition without the assignment or the terminator symbol. Referring back to the syntax above, the model component is simply P. In Fig. 1 the model component is the synchronisation between the labelled compositional definition pathway and the species E. As S and P already synchronise over alpha in the pathway definition, this synchronisation results in a reaction involving three species.

3 The Bio-PEPA Eclipse Plug-in

The syntax covered is that of the Bio-PEPA Eclipse Plug-in, one of two tools developed at Edinburgh University for the Bio-PEPA language. Built on top of the Eclipse platform [9], it offers an operating system agnostic editor and suite of time-series solvers for the construction and analysis of Bio-PEPA models. It provides the modeller with a rich modelling environment for Bio-PEPA, from the initial creation of the model through to the analysis and inspection of the results.

The features currently supported in the plug-in can be roughly grouped into those assisting the modeller in the writing of their models and features for the analysis of the model. Examples of the former include an editor (with Bio-PEPA-aware syntax highlighting) and static analysis of the model for detecting various syntactic errors. For the latter, the plug-in is capable of generating and displaying time-series results from ODE solvers or SSAs, with the option to export these results in both graphical and textual formats. Many of these features can be seen in Figure 2.

Once the model has been built, the plug-in parses the model and performs static analysis to detect a variety of modelling errors. If errors are present they are listed in the problems tab for the modeller to identify and correct. Once free from errors, the modeller has access to ODE solvers in the form of a Runge-Kutta implicit-explicit solver and a Dormand-Prince adaptive step-size solver [10], these being made available through the odeToJava library. The plug-in also supports analysis through stochastic simulators, with implementations of Gillespie's SSA [11], Gibson-Bruck [12] and the Tau-Leap algorithms [13] available. The ability to use different types of solvers has allowed us to discover errors in published modelling studies in computational biology [14].

The modeller also now has the ability to export the Bio-PEPA model in SBML format (level 2 version 3), with the hope to support other formats such as PRISM at a later point. This allows for other modelling tools to be utilised; in the case of SBML this allows access to applications such as COPASI or SBSI Visual, a CSBE initiative.

Instructions for downloading the Bio-PEPA Eclipse Plug-in are available from `http://biopepa.org` under Tools. From there instructions can be found for various versions of the Eclipse platform along with the URL for downloading the plug-in.

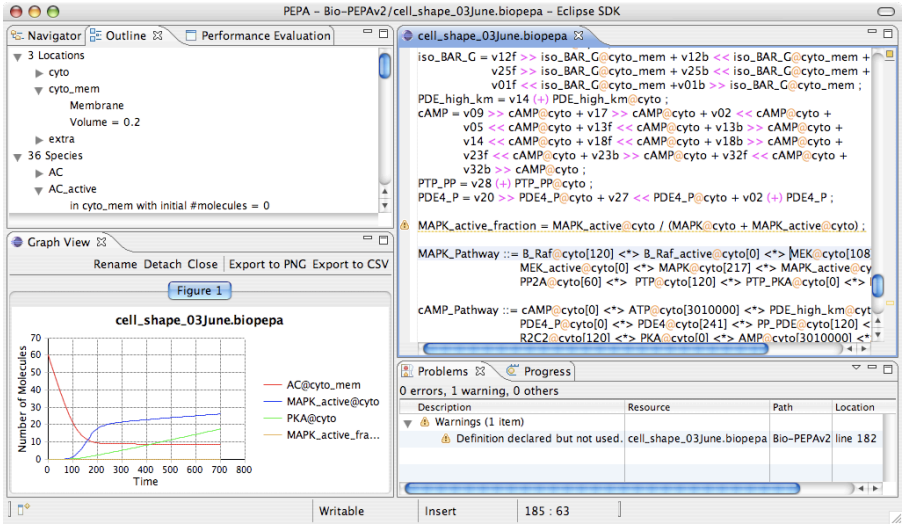


Fig. 2. Screen shot of the Bio-PEPA Eclipse Plug-in. The editor can be seen in the top left corner, with the outline view in the top right. The problems tab (bottom right) displays warnings and errors associated with the open models and the ability to plot results can be seen in the bottom left.

4 Conclusions

Here we have covered the syntax for the Bio-PEPA language, hopefully in sufficient detail to allow a modeller to use the plug-in. The different types of definitions have been described, detailing which parameters are required and which are optional. Finally, a brief overview of the plug-in has been given to highlight some of the key functionality currently present.

References

1. Ciocchetta, F., Hillston, J.: Bio-PEPA: A framework for the modelling and analysis of biological systems. Theoretical Computer Science **410**(33–34) (2009) 3065–3084
2. Akman, O., Ciocchetta, F., Degasperi, A., Guerriero, M.L.: Modelling Biological Clocks with Bio-PEPA: Stochasticity and Robustness for the *Neurospora Crassa* Circadian Network. In: CMSB 2009. LNCS (2009)
3. Ciocchetta, F., Duguid, A., Guerriero, M.L.: A compartmental model of the cAMP/PKA/MAPK pathway in Bio-PEPA. In: The 3rd Workshop on Membrane Computing and Biologically Inspired Process Calculi, Electronic Proceedings in Theoretical Computer Science (2009) To appear.
4. Guerriero, M.L.: Qualitative and Quantitative Analysis of a Bio-PEPA Model of the Gp130/JAK/STAT Signalling Pathway. TCSB special issue on Computational models for cell processes (2009)

5. Ciocchetta, F., Gilmore, S., Guerriero, M.L., Hillston, J.: Integrated simulation and model-checking for the analysis of biochemical systems. Electr. Notes Theor. Comput. Sci. **232** (2009) 17–38
6. Ciocchetta, F., Guerriero, M.L.: Modelling biological compartments in Bio-PEPA. Electronic Notes in Theoretical Computer Science **227** (2009) 77–95
7. Heimsund, B.: Matrix toolkit for java. (`http://code.google.com/p/matrix-toolkits-java/`)
8. Kwiatkowska, M., Norman, G., Parker, D.: PRISM: Probabilistic symbolic model checker. In Field, T., Harrison, P.G., Bradley, J., Harder, U., eds.: Computer Performance Evaluation: Modelling Techniques and Tools. Number 2324 in Lecture Notes in Computer Science, London, UK, Springer (2002) 200–204
9. : Eclipse website. (`http://www.eclipse.org`)
10. Stoer, J., Bulirsch, R.: Introduction to Numerical Analysis. Springer-Verlag (1993)
11. Gillespie, D.T.: Exact stochastic simulation of coupled chemical reactions. Journal of Physical Chemistry **81**(25) (1977) 2340–2361
12. Gibson, M.A., Bruck, J.: Efficient exact stochastic simulation of chemical systems with many species and many channels. Journal of Physical Chemistry **104** (2000) 1876–1889
13. Gillespie, D.T., Petzold, L.R.: Improved leap-size selection for accelerated stochastic simulation. Journal of Chemical Physics **119**(16) (2003) 8229–8234
14. Calder, M., Duguid, A., Gilmore, S., Hillston, J.: Stronger computational modelling of signalling pathways using both continuous and discrete-state methods. In Priami, C., ed.: CMSB. Volume 4210 of Lecture Notes in Computer Science., Springer (2006) 63–77

Efficient compositional simulation of circadian models using Bio-PEPA

Stephen Gilmore and Konstantinos Markakis
School of Informatics
The University of Edinburgh

August 19, 2009

Abstract

We describe a novel method of performing efficient stochastic simulation of models of biological networks with time-dependent behaviour (such as circadian clock models). Networks are described in the Bio-PEPA process algebra and then compiled to sets of coupled simulation models, each instantiated with different rate parameters. These models have simpler kinetic functions than those found in a single model, allowing the application of more efficient simulation methods.

1 Introduction

Circadian clock models are important in computational biology to help us understand how living organisms from plants to humans react to changes in the night/day cycle. However, they are complex to simulate because of the difficulty of encoding the switch from day to night conditions in the simulation. The phenomenon of increased activity during daylight hours (due to higher temperature and the presence of light) is often encoded in ODE models using a conditional construct in the kinetic laws. The intention is to write a construct of the following form for a rate expression in a kinetic law for reaction r:

$$\textbf{if } day \textbf{ then } k_d \textbf{ else } k_n$$

where k_d is a kinetic constant describing the rate of reaction r in the daytime and k_n is a kinetic constant describing the rate of reaction r in the nighttime. Since ODE solvers have a built-in notion of time, but not of day and night, then the predicate *day* usually needs to be implemented through reference to a *time* variable (modulo 24 hours). If the hours of daylight are from 6a.m. to 6p.m. the expression then becomes:

$$\textbf{if } (time\%24) \geq 6 \wedge (time\%24) < 18 \textbf{ then } k_d \textbf{ else } k_n.$$

Since ODE solvers and chemical kinetics simulators do not always allow conditional constructs in kinetic laws the conditional often needs to be replaced by the use of a discontinuous θ function[1] which returns 1 for *true* and 0 for *false* leading to an expression such as the following.

$$\big(\theta((time\%24) \geq 6 \wedge (time\%24) < 18) * k_d\big) + \big(\theta((time\%24) < 6 \vee (time\%24) \geq 18) * k_n\big)$$

Encoding conditional expressions in this way has several disadvantages:

1. the expressions in the kinetic laws become complex and hard to read;
2. the kinetic laws make explicit reference to the current simulation time; and
3. the kinetic laws are no longer differentiable because of the use of the discontinuous θ function.

[1] Or sometimes a smoothly differentiable function which approximates this.

The first of these is a problem because we would like our models to be as easy to understand as possible (to reduce the potential for modelling errors). The second disadvantage is a problem for Gillespie simulation frameworks which do not usually expose the current simulation time to the kinetic functions (because for a Gillespie simulation kinetic functions must depend only on rate constants and molecule counts). The third of these disadvantages prevents modellers from deriving the Jacobian matrix of derivative functions which can be used to improve the speed, accuracy and robustness of a numerical integrator. Without a representation of the Jacobian matrix a numerical integrator typically approximates by using finite differences. However, this approximation will not always be usable in practice because it may require the maximum allowed relative error from one time point to the next to be decreased to an unacceptably small value. Finally, without a Jacobian matrix efficient approximate stochastic simulation algorithms such as Gillespie's τ-leap [1] cannot be used at all.

However, none of these problems arise if we express our model description as a pair of coupled models, one using the set of daytime kinetic parameters (k_d) and the other using the set of nighttime kinetic parameters (k_n). In this setting:

1. the expressions in the kinetic laws are simplified;
2. the kinetic laws do not need to make reference to the simulation time; and
3. the kinetic laws need not make use of a θ function and remain differentiable.

Maintaining a pair of closely related models would of course be a maintenance headache in practice but using the Bio-PEPA process algebra [2] as the high-level modelling language for the problem we can generate this pair of coupled models. In this way we decompose the modelling problem and we can perform separate simulations which can be recombined to give the desired results. We have provided this functionality for Bio-PEPA by adding the Bio-PEPA Workbench to the Dizzy simulation framework [3].

2 The Bio-PEPA Workbench

The Bio-PEPA Workbench[2] is a command line software tool which provides a platform for analysis of Bio-PEPA models using various techniques. This is accomplished by parsing a text file which contains a model written in the Bio-PEPA language and capturing the model in an intermediate tree-like structure. This intermediate representation is then used as a template for producing several outputs, using external tools. These outputs include:

- A differential equation model, in terms of a high level "vector field" representation. This representation is used by VFGen which in turn generates ODE models suitable for analysis by the Sundials ODE suite and MATLAB.
- A stochastic simulation model in C++ code which is used by stochkit, a software toolkit that features an implementation of Gillespie's SSA. The results generated by stochkit are then plotted with gnuplot.
- A Continuous Time Markov Chain model expressed in the reactive modules language, understood by the PRISM model checker, and a CSL formulae file corresponding to than model.
- A LaTeX report (along with the corresponding PDF) and an HTML page containing a formatted version of the Bio-PEPA model and the graphs that were generated from the simulation runs.
- A translation of the Bio-PEPA model to the CMDL language, as used by the Dizzy simulator.

The file containing the Bio-PEPA model definition does not include arithmetic values assigned as initial concentrations of species or rate constants. Instead, these values are parsed from a separate CSV file which is a part of the model definition. The CSV file consists of a row containing the names of the species and rate constants of the model, and a number $n \geq 1$ rows containing comma-separated values corresponding to the initial concentrations and rate constants. This gives the capability to perform multiple analysis of the same model based on several sets of initial concentration and rate constant values. The Bio-PEPA workbench is written in Standard ML, and compiled to Java bytecode using the MLj compiler[3].

[2] http://homepages.inf.ed.ac.uk/stg/software/biopepa
[3] http://www.dcs.ed.ac.uk/home/mlj

3 Integration of the Bio-PEPA Workbench into Dizzy

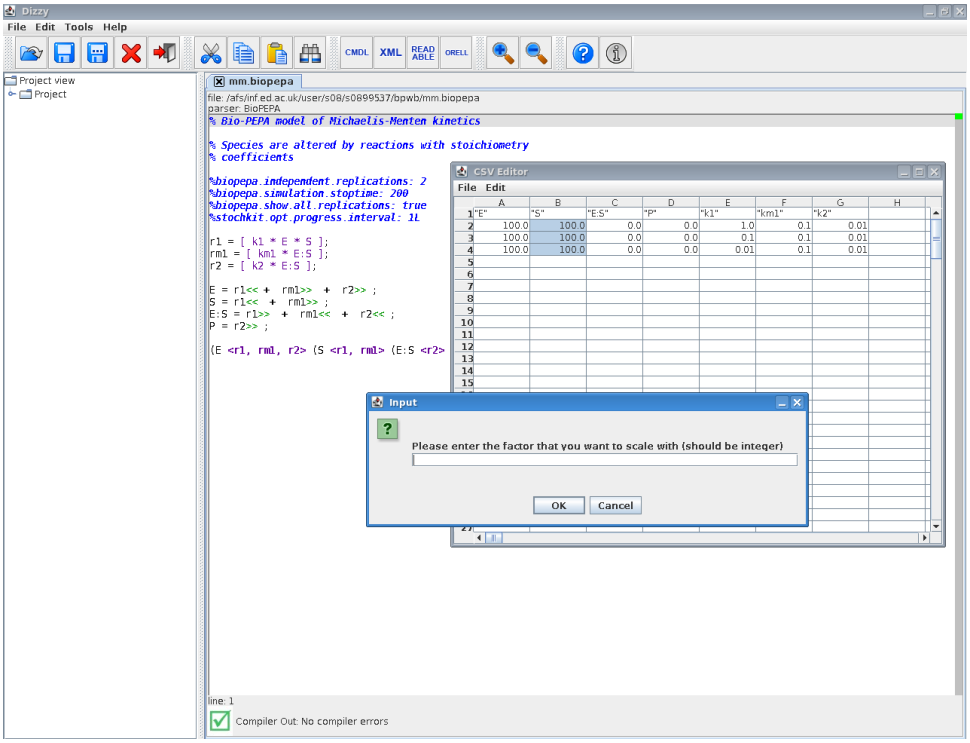
Dizzy features a modular software framework in many aspects. New simulators and new parsers for model languages can be implemented and added easily through well-defined interfaces. Bio-PEPA could indeed, be implemented and integrated using the relevant interface, which is called `IModelBuilder` and it defines a method for accepting an `InputStream` object and returning a `Model` object. The `Model` class facilitates a unique and consistent representation of a biochemical network in Dizzy. It includes objects representing reactions, species (divided into reactants and products), stoichiometric coefficients, concentrations etc. Each one of the description language parsers, performs the translation and capture of the underlying model into a `Model` object, which is used for the simulation. In order to implement the `IModelBuilder` interface, however, we should have a parser for Bio-PEPA implemented natively in Java. Instead we developed a new class which extends Dizzy's `EditorPane` and overrides the `processModel()` method. This method passes the text from the Dizzy text area into an appropriate translator class (according to the model language which is currently enabled on Dizzy) and returns a `Model`. The overridden `processModel()` method is responsible for passing the Bio-PEPA model text entered into the text area to an internal method which performs the intermediate steps in order to translate the Bio-PEPA model into a number of Dizzy models (according to the number of sets of CSV values) by calling the newly created runtime library. Then it brings up a GUI chooser from which the user can select one of the output models to simulate. The attribute `mEditorPane` is assigned either an `EditorPane` object or a `BioPEPAPane`, according to what type of model is currently displayed in the text area. The intermediate steps of compilation from a Bio-PEPA text model to a `Model` are now captured by `processModel()` of `BioPEPAPane` which is called instead, when `mEditorPane` is assigned a `BioPEPAPane` object, using virtual method invocation. All other calls made to `processModel()` will end up calling the overridden method.

4 Simulating a Bio-PEPA model

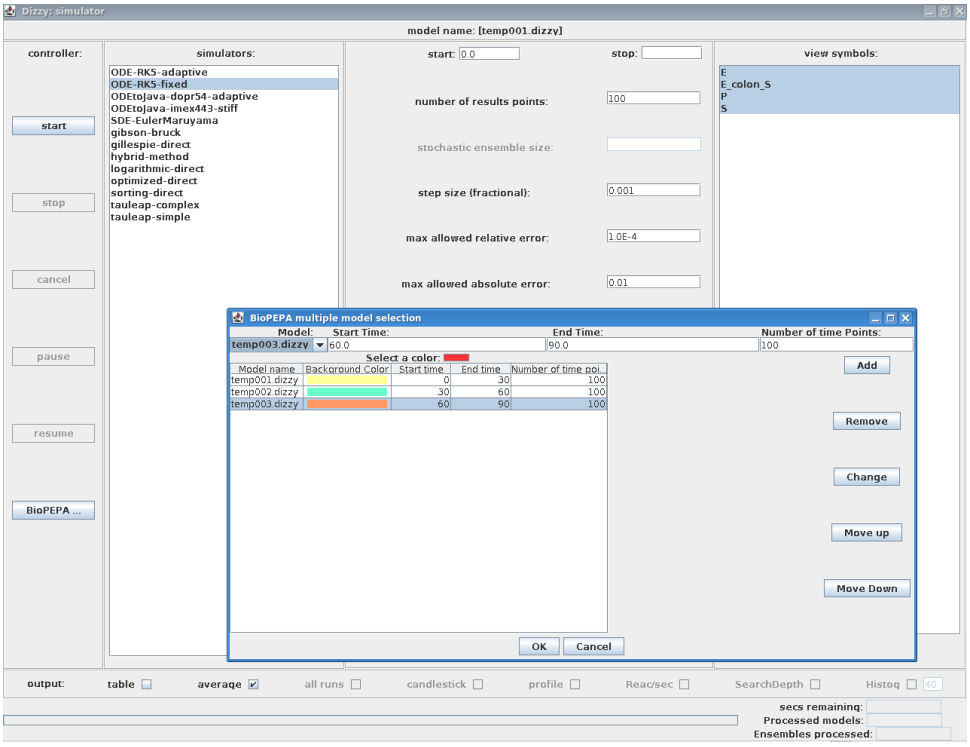
An interesting characteristic of representing a biochemical network as a Bio-PEPA model is the separation of the initial concentrations and parameter values from the semantics of the model. This abstraction enables the Bio-PEPA Workbench to perform multiple simulation runs for each set of values in the CSV file. The final constituent of the extension involves transferring this capability to the Dizzy tool. It can be divided into three parts:

- Extension of the simulation framework to support successive simulations of multiple models for complementary time intervals.
- Implementation of new graph outputs to support colour marking of the graphs, according to the model running for particular time ranges.
- Creation of a GUI component for choosing the model that will run for particular time intervals, the number of time points that will be simulated for each time interval, and the colour that will be used for marking the graph according to the model and the time interval.

The extension offers the capability of running multiple simulations of a Bio-PEPA model based on the CMDL outputs which are acquired from the compilation of the Bio-PEPA model from the integrated library and the different sets of values in the CSV file. For example, assuming that there are three sets of values in the CSV file, the Bio-PEPA Workbench outputs three CMDL models with one set of values assigned to each. The modeller has the ability to choose time intervals which will be assigned to each of the models, with the restriction that the intervals must be exhaustive and non-overlapping. In the original version of Dizzy, a call to the `processModel()` method in `EditorPane` would perform the translation from a model written in CMDL, to a `Model` object which is an independent representation of the model, regarding the description language used, ready to simulate. Afterwards, the object would be passed to the simulator framework. The simulation framework is responsible for manipulating the numerics (i.e. concentrations of species) of the `Model` object during the simulation process. The extension requires that all `Model` objects created from the Bio-PEPA model and the CSV file should be passed to the extended simulation framework, thus the first step was to add a method to `BioPEPAPane` in order to process the CMDL files and return all `Model` objects. The second step of this part of the extention involved extending the functionality of the simulation framework in order to process all models with one of the simulation algorithms of Dizzy in one simulation run.



(a)



(b)

Figure 1: (a) The spreadsheet component as a part of Dizzy's functionality; and (b) the GUI component featuring the selection of the models, time-intervals for simulation and colours for the output graphs.

5 Simulation management

To manage the more complex simulation process which involves coordinating multiple models, a new class extending `SimulationLauncher` was created, namely `SimulationLauncherBioPEPA`. According to the description language which is enabled at the point of request for simulation, the respective class will be used as the simulation GUI. The methods that are overridden in the new class, are the ones responsible for the simulation process and the visualisation of the results. `SimulationLauncherBioPEPA` constitutes the extended simulation framework and contains a constructor which accepts multiple models. The ability to select which models are going to be simulated for particular time intervals is carried out though a new class which is instantiated from the extended simulation framework. This class provides a GUI component which enables the user to choose the models, the corresponding time-intervals, the number of simulated time points for each model and the marker colours for the selected models. Figure 1(b) shows the component as a part of the simulation GUI. An added feature permits the user to repeat selected elements for a number of times by adding periodicity to the list. This class also contains methods for checking the inputs in the text fields. More specifically, the time-intervals must be exhaustive and non-overlapping. The overridden `runSimulation()` method in class `SimulationLauncherBioPEPA` performs the necessary steps for the correct simulation of multiple models. An important assumption had to be made in order to proceed to the implementation. In each simulation time step, the resulting concentration value of a species depends on the model that is currently running, the simulation algorithm and the concentration value of the previous time step. This is a valid assumption because each time step can be considered as the starting time for the simulation of the model. The deterministic simulation algorithms will always behave in a predetermined way according to the initial concentration, while the stochasticity in the stochastic algorithms does not depend on time. Another reason that the assumption remains valid is that time-specific events are not implemented in the current extension. Therefore there is no discontinuity in the concentration values during a simulation.

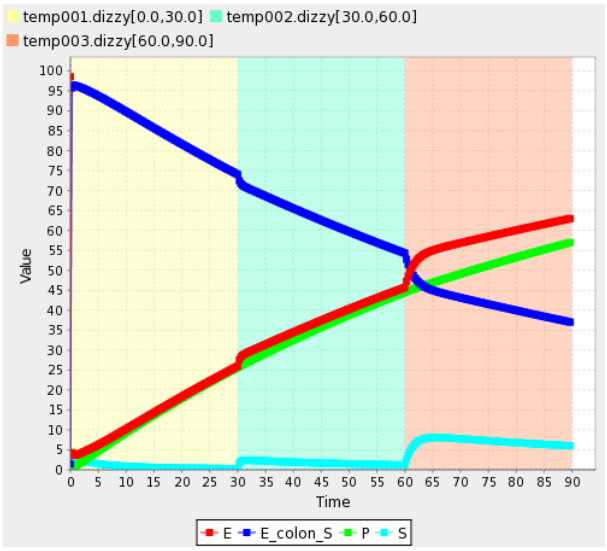


Figure 2: The output graph resulting from the simulation of the Michaelis-Menten kinetics model for different parameter values.

6 Conclusions

We have implemented a simulation management framework for the Bio-PEPA language which is well-suited to supporting the modelling and analysis of clocked models. The time series is divided into phases with a set of parameter values assigned to each phase. Phases need not be of equal length, and there many be any number of them (not just two). A cycle of simulation models is generated and simulated in isolation for each phase of the cycle. The results are combined to give a presentation of the results across several phases of the cycle.

The use of an explicit time variable or a discontinuous function is avoided, making it possible to use efficient numerical integrators for continuous simulation or efficient stochastic simulators such as τ-leap. Thus both contin-

uous and deterministic simulation results are obtained more quickly than when using an approach which encodes the phase-change using a θ function.

The method is implemented as an extension to the Dizzy simulator which includes a copy of the Bio-PEPA Workbench tool. A syntax-highlighting editor for Bio-PEPA is provided, together with a built-in Java spreadsheet for editing the files of parameter data read by the Bio-PEPA Workbench when generating Dizzy models for simulation. The compilation from Bio-PEPA to Dizzy is completely automated by the workbench, allowing Dizzy models to be generated without user intervention.

The simulation process is driven by a user interface component which allows users to specify phases of the simulation and to set the periodicity of the repetition. This flexible design allows for initial phases of the simulation which are not repeated. For example, a simulation of a 12-hour day might begin with six hours of night, thereafter followed by alternations of twelve hours of daylight and twelve hours of night, for any number of repetitions up to a finite time horizon.

The simulation process is driven by a single source model in the Bio-PEPA process algebra, avoiding the need to keep modified copies of a low-level simulation, and avoiding the need to ensure that these are consistent when run.

Acknowledgements: Stephen Gilmore is supported by the EPSRC grant EP/E031439/1 "Stochastic Process Algebra for Biochemical Signalling Pathway Analysis".

References

[1] Daniel T. Gillespie. Approximate accelerated stochastic simulation of chemically reacting systems. *J. Comp. Phys.*, 115(4):1716–1733, 2001.

[2] Federica Ciocchetta and Jane Hillston. Bio-PEPA: A framework for the modelling and analysis of biological systems. *Theoretical Computer Science*, In Press, Corrected Proof, 2009. To appear.

[3] S. Ramsey, D. Orrell, and H. Bolouri. Dizzy: stochastic simulation of large-scale genetic regulatory networks. *J. Bioinf. Comp. Biol.*, 3(2):415–436, 2005.

On the Formalisation of Gradient Diffusion Models of Biological Systems

Andrea Degasperi and Muffy Calder

Department of Computing Science, University of Glasgow
Glasgow G12 8QQ, Scotland, UK
{andrea,muffy}@dcs.gla.ac.uk

Abstract

Many formal models for biological systems include a notion of topological space in the form of compartments. We consider the problem of modelling gradient diffusion systems that require a notion of metric space. We define diffusive *slots*, which govern local interactions and the diffusion from and to adjacent slots, and *areas*, which comprise one or more slots. We propose that a generic formalism is not suitable for modelling gradient diffusion systems, rather we tailor formalisms to particular scenarios, e.g. to biological systems with a given shape. An example of diffusion of nitric oxide in blood vessels illustrates the approach.

1 Introduction

Numerous languages and frameworks for discrete representation and reasoning about biological interactions have been developed over the last decade. The main contributions have been in the area of *process algebras* [15, 14, 9, 4, 3, 7], where a biochemical system is modelled by interacting processes representing molecules or species; *rewriting rules* [8, 2], where a system is modelled by molecules and their binding sites and state change is governed by rules that rewrite parts of the system; and *high level languages* [5, 13], where a biochemical system is specified by a list of parameters, reactions and additional grammatical features, which can be converted to any other formalism.

Here, we consider the problem of representing and reasoning about *space* in a discrete formalisation and in particular, we consider *gradient diffusion systems*. While the formalisms above have been extended to include a notion of space, it is usually by means of a topological space with *compartments* that are private locations, each of which is governed by a specific set of reactions and between which interactions are prohibited. Compartments are delimited by membranes that are static or dynamic, i.e. they change through time, leading to changes in the configuration of the compartmental structure.

In models of gradient diffusion systems, so far formalised mainly by *cellular automata* [1], notions like position and distance between molecules are required. These models apply, for example, to biological systems where proteins are translated in specific areas of tissues or

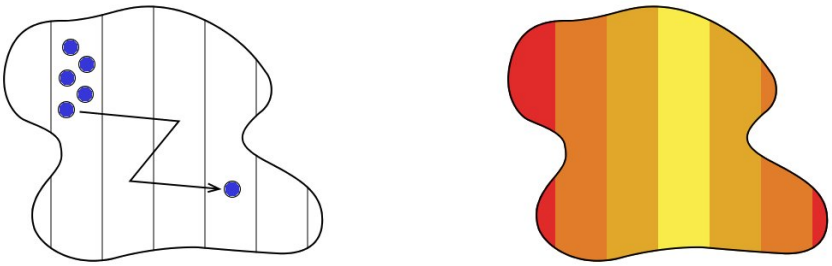


Figure 1: Two spatial approaches: single unit tracking (on the left) and average unit amount in an area (on the right).

organs. The diffusion distance, i.e. the ability of proteins to migrate far from the source, is fundamental for the correct functioning of the system: very precise phenotypes might be connected to areas where different proteins meet. An example of this phenomenon is pattern formation during morphogenesis of the Drosophila embryo [12].

The paper is organised as follows. In the next section we consider modelling choices for spatial models and in section 3 we propose a framework for the formalisation of gradient diffusion models. In section 4 we give an overview of a gradient diffusion model example: diffusion of nitric oxide in blood vessels. Our conclusions and future work are in section 5.

2 Modelling choices for space

In this section we give an overview of modelling choices for the formalisation of biological systems when a notion of location is required.

Individual and population view: in formal models of biochemical interactions, an elementary unit is either a single molecule, a fixed number of molecules, or a fixed amount of concentration. Formalising these elementary units in space, one has a choice between labelling

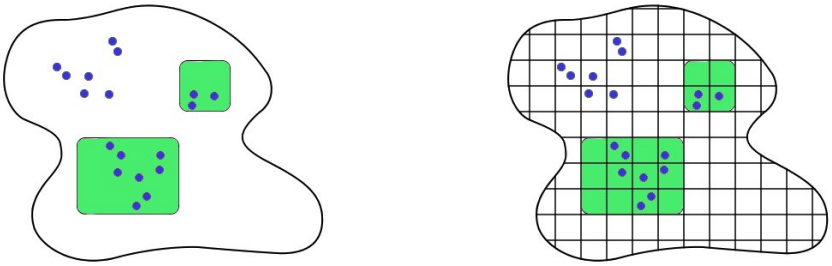


Figure 2: Compartments can be extended with a notion of position.

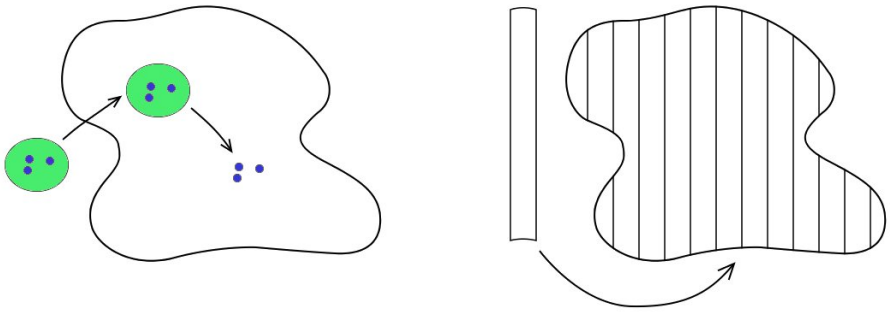


Figure 3: Modification of the compartmental structure or of the spatial shape.

a unit with its position, or simply tracking the number of units of the same type that are at a specific position. This is illustrated in Figure 1.

Many formalisms, such as Bioambients [16], Brane Calculi [6], bioκ-calculus [11] and stochastic bigraphs [10], although capable of implementing the individual view, choose the population view, primarily to reduce the state space explosion problem. An example of a process algebra based on the population view is Bio-PEPA [7].

Topological and metric space: as previously mentioned, compartments delimit areas where different molecules or different interactions take place. This is often suitable when considering a cell: typical compartments are cytoplasm and nucleus. But in diffusion systems, one often considers tissues or organs, at an higher organisational level. The notion of compartment is no longer sufficient. One needs to distinguish between diffusive *slots*, which govern local interactions and the diffusion from and to adjacent slots, and *areas*, usually comprising one or more slots, which encompass the delimiting function of compartments. This is illustrated in Figure 2. Models that include a notion of diffusion in a metric space are usually continuous and based on or derived from partial differential equations (PDEs). We note that in these models, the shape of the biological entity being modelled and the coordinate system used are critical factors.

Compartments and shape modification: some of the above mentioned formalisms, e.g. Bioambients, Brane Calculi and stochastic bigraphs, allow compartment manipulation. When the shape and the position of a biological entity is taken into consideration, other modifications might be necessary, such as the addition of new slots or the reassignment of a slot to a different area. This is illustrated in Figure 3. Examples of when these additional modifications might be required are tumour growth and organ morphogenesis.

Clearly a form of metric space is required for modelling gradient diffusion. Our preliminary investigations have led us to believe that a generic formalism will not be useful because there is huge variety of overall shape of the biological entity, and consequently the chosen coordinate system. We therefore concentrate on particular scenarios, as defined in the next section.

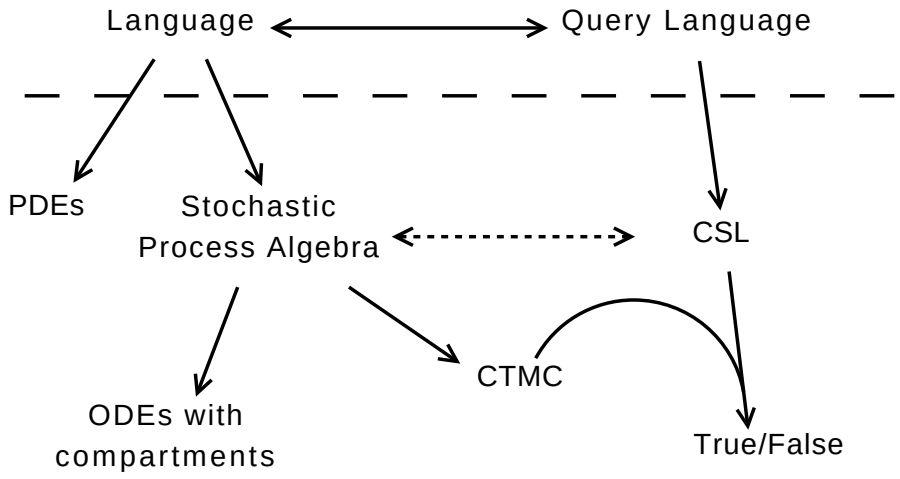


Figure 4: Translation and query for a description of a given scenario

3 High level languages for specific scenarios

We define a *scenario* as a biological system and a set of assumptions about shape, compartmentalisation and equations governing diffusion. The difference between models of a scenario will be in the level of detail, the types of molecules and biochemical interactions.

In order to formalise gradient diffusion models, we propose a high level descriptive language that is designed specifically for a single scenario. These are the main advantages:

Improved readability and maintenance: a scenario specific language is compact, it states only the information that distinguish models belonging to the same scenario. Notions and assumptions that are shared by models of the same scenario do not need to be stated explicitly in the language. Consequently descriptions are easy to read, write and maintain by modellers that are familiar with the scenario.

Modularity and translations: a formal language can be parsed and translated to other formalisms, such as process algebras or rewriting rules, if these are suitable for the scenario. The translation is automatic and ensures reliability of model formulation. If new mathematical models or formalisms that are more suitable are later defined, then new translations can be used while the high level language is unaltered.

High level queries: it is possible to formulate queries based on the high level language. Results are computed depending on the underlying formalism.

4 Example: nitric oxide bioavailability in blood vessels

In this section we briefly present an example scenario: diffusion in blood vessels.

Consider modelling nitric oxide (NO) bioavailability in blood vessels. Models of this scenario aim to determine the diffusion distance of NO along the radius of a vessel, where NO is produced in a narrow region on the internal wall of the vessel. Numerous models have been developed over the last decade (see [17] for a complete review) and almost all of them share

the same assumptions and use the same diffusion governing equations. In particular, a vessel is modelled as a cylinder with partial differential equations (PDEs), using Fick's law of diffusion in cylindrical coordinates. Compartments define areas such as endothelium (where NO is produced), vascular wall, and lumen (i.e. where the blood flows). Another common assumption is that the diffusion operates only in the radial direction, while it can be considered negligible in other directions.

We have defined a high level language from which both a traditional PDE and a stochastic process algebra (SPA) model can be derived. The SPA in this case is Bio-PEPA with static compartments: the space (in this case the radius) has to be divided into a number of slots defined by the modeller. The Bio-PEPA model is derived according to the implicit assumptions of the scenario and to information in the high level description. This means that the characteristics of each compartment, given by the rates of transport between compartments, the volume and the associated reactions, are derived automatically. From the Bio-PEPA description thus derived, other modelling approaches become accessible, such as ordinary differential equations (ODEs) with compartments, and continuous time markov chains (CTMCs with levels) (see [7] for details). Finally, a high level query language can be mapped to continuous stochastic logic (CSL), for reasoning about the states of the chain. A schematic representation of these translations is given in Figure 4.

5 Conclusions and Future Work

We have considered the problem of modelling gradient diffusion systems requiring a notion of metric space. In order to model space, we introduced diffusive *slots*, which govern local interactions and the diffusion from and to adjacent slots, and *areas*, comprising one or more slots, which encompass the function of compartments. We outlined an approach for modelling based on a high level language description for a given scenario and gave a brief overview of modelling an example gradient diffusion system: bioavailability of nitric oxide in blood vessels.

Future work includes formal proof the relationship between the underlying models, i.e. between PDEs, ODEs and CTMCs with levels, and further investigation of suitable query languages for gradient diffusion models.

References

[1] O. L. Bandman. Comparative Study of Cellular-Automata Diffusion Models. *PaCT-99, LNCS 1662*, pages 395-409, 1999.

[2] M. L. Blinov, J. R. Faeder, B. Goldstein and W. S. Hlavacek. BioNetGen: software for rule-based modeling of signal transduction based on the interactions of molecular domains. *Bioinformatics*, Vol. 20, no. 17, pages 3289-3291, 2004.

[3] L. Bortolussi. Stochastic concurrent constraint programming. *Proceeding of QAPL2006: 4th International workshop on quantitative aspects of programming languages*, 164:65-80, 2006.

[4] M. Calder, S. Gilmore, and J. Hillston. Modelling the influence of RKIP on the ERK signalling pathway using the stochastic process algebra PEPA. *T. Comp. Sys. Biology*, VII, volume 4230 of LNCS, pages 1–23, Springer, 2006.

[5] L. Calzone, F. Fages, S. Soliman. BIOCHAM: An environment for modelling biological systems and formalizing experimental knowledge. *Bioinformatics*, 22, pages 1805-1807, 2006.

[6] L. Cardelli. Brane Calculi. *CMSB 2004*, LNCS (LNBI), vol. 3082, pp. 257-278. Springer, Heidelberg, 2005.

[7] F. Ciocchetta, and J. Hillston. Bio-PEPA: an extension of the process algebra PEPA for biochemical networks. Proc. of *FBTC 2007*, volume 194/3 of ENTCS, pages 103–117, 2008.

[8] V. Danos, J. Feret, W. Fontana, R. Harmer and J. Krivine. Rule-based modelling of cellular signalling. *Proceeding of the 18th International Conference on Concurrency Theory (CONCUR'07)*, LNCS, Sep 2007.

[9] J. Hillston. *A Compositional Approach to Performance Modelling*, Cambridge University Press, 1996.

[10] J. Krivine, R. Milner and A. Troina. Stochastic Bigraphs. *ENTCS* 218 , pages 73-96, 2008.

[11] C. Laneve, F. Tarissan. A simple calculus for proteins and cells. *ENTCS* 171, pages 139-154, 2007.

[12] C. M. Mizutani, Q. Nie, F. Y.M. Wan, Y. Zhang, P. Vilmos, R. Sousa-Neves, E. Bier, J. L. Marsh and A. D. Lander. Formation of the BMP Activity Gradient in the Drosophila Embryo. *Developmental Cell*, Vol. 8, 915924, June, 2005.

[13] M. Pedersen, G. Plotkin. A language for biochemical systems. *CMSB 2008*.

[14] C. Priami and P. Quaglia. Beta-binders for biological interactions. Proc. of *CMSB'04*, Volume 3082 of LNCS, pages 20–33, Springer, 2005.

[15] A. Regev. Representation and simulation of molecular pathways in the stochastic π-calculus. *Proceedings of the 2nd workshop on Computation of Biochemical Pathways and Genetic Networks*, 2001.

[16] A. Regev, E. Panina, W. Silverman, L. Cardelli and E. Shapiro. Bioambients: An abstraction for biological compartments. *Theoretical Computer Science*, 325(1), pages 141-167, 2004.

[17] N. M. Tsoukias. Nitric Oxide Bioavailability in the Microcirculation: Insights from Mathematical Models. *Microcirculation*, 15:8, 813–834, 2008.

Modelling the bubonic plague in a prairie dog burrow, a work in progress

Soufiene Benkirane, Carron Shankland, Rachel Norman, Chris McCaig
University of Stirling

1 Introduction

Plague is caused by a bacteria known as *Yersinia pestis*, and is affecting over 200 mammalian species worldwide [6], including human. It is famous for being the Black Death, who killed millions of people in Europe. Even now, the disease is still a serious health problem in some parts of the world [3], and it is lethal without immediate medicale attention.

In North America, the black-tailed prairie dog is a keystone species in the flora and fauna dynamics, with over a hundred related species [5]. However, the introduction of this exotic disease has severe consequences on the population of prairie dogs, and thus affects the whole ecosystem. Indeed, prairie dogs are extremely susceptible to the plague, and their mortality rate is almost 100% when infected. The main source of infection has long been thought to be the prairie dog flea (*Oropsylla hirsuta*)[8]. However, a recent paper [9] suggested that the flea might not be the main vector of infection, but that direct transmission and infected soil might have a more important role. This paper tries to reproduce the results obtained in Webb et al. [9], while using a PEPA model instead of Ordinary Differential Equations (ODEs). The main objective is to determine if PEPA is capable of modelling an epidemiological model, and what issues are raised while doing so. The bubonic plague is particularly suitable for addressing this issue, as it requires several different features to be added to describe the disease behaviour.

2 Presentation of the disease behaviour

2.1 The bubonic plague behaviour

The disease has three main vectors of infection. The first is direct transmission. In other words, if a prairie dog affected by the bubonic plague meets one that is not, there is a chance that the bubonic plague is transmitted to the non-infected prairie dog. The second vector is indirect transmission. In this case, an infectious prairie dog infects the soil, either with its faeces or when it dies, its body still being very infectious. The final vector of infection is the flea. A flea

$$
\begin{aligned}
S &\stackrel{def}{=} (contact\_direct, \top).Exposed + (indirect\_contact, \top).Exposed + \\
&\quad (contact, \top).Exposed(birth, birthrate).S + (die, death\_rate\_PD).Dead + \\
&\quad (infantdeath, \top).SDying \\
SDying &\stackrel{def}{=} (dieNewBorn, \top).Dead \\
Exposed &\stackrel{def}{=} (contact\_direct, \top).Exposed + (indirect\_contact, \top).Exposed + \\
&\quad (contact, \top).Exposed + (infected, incubation\_rate).Inf + \\
&\quad (die, death\_rate\_PD).Dead + (infantdeath, \top).Exposed \\
Inf &\stackrel{def}{=} (contact\_direct, \top).Inf + (infect\_flea, inf\_flea\_rate).Inf + \\
&\quad (indirect\_contact, \top).Inf + (infect\_environment, inf\_env\_rate).Inf + \\
&\quad (contact, \top).Inf + (die\_inf, death\_rate\_inf\_PD).Dead + \\
&\quad (die, death\_rate\_PD).Dead + (infantdeath, \top).Inf \\
Dead &\stackrel{def}{=} (indirect\_contact, \top).Dead + (alive, \top).S + (infantdeath, \top).Dead \\
SainPD &\stackrel{def}{=} (die, \top).DeadPD + (infantdeath, birthrate * totalNbPD/k).SainPD + \\
&\quad (infected, \top).InfectedPD + (dieNewBorn, \top).DeadPD \\
InfectedPD &\stackrel{def}{=} (die\_inf, \top).DeadPD + (die, \top).DeadPD + \\
&\quad (contact\_direct, contact\_direct\_rate).InfectedPD \\
DeadPD &\stackrel{def}{=} (birth, \top).TempPD \\
TempPD &\stackrel{def}{=} (alive, big).SainPD \\
Sq &\stackrel{def}{=} (death, fdeathrate).DeadF + (findHost, a * meanNbHost).Sh \\
Sh &\stackrel{def}{=} (flea\_birth, fbirthrate).Sh + (death, fdeathrate).DeadF + \\
&\quad (leaveHost, leaveHostRate).Sq + (infect\_flea, \top).Eh \\
Eq &\stackrel{def}{=} (death, fdeathrate).DeadF + (findHost, a * meanNbHost).Eh + \\
&\quad (infectiousF, flea\_incubation\_rate).Iq \\
Eh &\stackrel{def}{=} (flea\_birth, fbirthrate).Eh + (death, fdeathrate).DeadF + \\
&\quad (leaveHost, leaveHostRate).Eq + (infect\_flea, \top).Eh + \\
&\quad (infectiousF, flea\_incubation\_rate).Ih \\
Iq &\stackrel{def}{=} (death, finfdeathrate).DeadF + (findHost, a * meanNbHost).Ih \\
Ih &\stackrel{def}{=} (death, finfdeathrate).DeadF + (leaveHost, leaveHostRate).Iq + \\
&\quad (infect\_flea, \top).Ih + (contact, transmission\_rate * a * meanNbHost).Ih \\
DeadF &\stackrel{def}{=} (flea\_alive, \top).Sq \\
ReservoirFlea &\stackrel{def}{=} (death, \top).DeadFlea \\
DeadFlea &\stackrel{def}{=} (flea\_birth, \top).TempFlea \\
TempFlea &\stackrel{def}{=} (flea\_alive, big).ReservoirFlea \\
Infenv &\stackrel{def}{=} (indirect\_contact, contact\_env\_rate * totalNbPD).Infenv + \\
&\quad (decay, decay\_rate).Noninfenv \\
Noninfenv &\stackrel{def}{=} (infect\_environment, \top).Infenv
\end{aligned}
$$

$$
\begin{aligned}
&((S[98] \parallel Inf[2] \parallel Dead[100]) \underset{\{*\}}{\bowtie} (SainPD[98] \parallel InfectedPD[2] \parallel DeadPD[100])) \\
&\underset{\{*\}}{\bowtie} \\
&((Sq[990] \parallel Iq[10] \parallel DeadF[10000]) \underset{\{*\}}{\bowtie} (ReservoirFlea[1000] \parallel DeadFlea[10000])) \\
&\underset{\{*\}}{\bowtie} Noninfenv[100000]
\end{aligned}
$$

Figure 1: The bubonic plague in prairie dogs PEPA model.

can be infected by an infectious prairie dog while feeding on him. Once inside the flea's body, the bacteria blocks the proventriculus [1], preventing the flea from eating. As it cannot manage to eat anymore, the flea starts biting more often. If it goes to a different prairie dog, it might infect it. The flea finally starves to death after a few days.

2.2 The model

In order to model the disease, an SI model [4] will be used. In these models, the population is divided in two: the *Susceptibles* and the *Infectious*. The *Susceptibles* are individuals that do not have the disease. They have a chance to become *Exposed* if they are contacted by an *Infectious*. After the incubation period, they themselves become infectious. So an infectious individual can spread the disease. In the case of the bubonic plague, the *Infectious* cannot recover, the disease is deadly in almost 100% of cases [9, 7].

In the case of the bubonic plague in the prairie dog population, direct transmission, transmission via fleas, and transmission via the infected soil had to be modelled. The model that has been used in this study is presented in Figure 1. It is divided into five section: prairie dogs, a mirror image of prairie dogs, fleas, a mirror image of fleas and the soil. Prairie dogs can be in five different states: *S*, *SDying*, *Exposed*, *Inf* or *Dead*. This corresponds to the states describing an SI model. However, two states can raise questions. *SDying* is used in order to model density dependent birth. Indeed, in this model, birth occurs at a constant rate *birthrate*. However, some of the infants can die, because the density of the population is too high. This results in an overall population increase of $birthrate \times S(1 - S/K)$ which is very similar to the term used by Webb et al. [9] $birthrate \times S(1 - N/K)$ (with N the total population of prairie dogs) [1]. The second unusual state is *Dead*. This state is necessary as it is not possible in PEPA to create new sequential components, so these represent ghosts, or potential newborns. The second section, the mirror image of the prairie dogs' section, is necessary to model all the interactions between prairie dogs, such as direct transmission of the disease or birth. This is explained in more details by Benkirane et al. [2]. The third and fourth sections detail flea behaviour, in a similar fashion to prairie dogs. Finally, the fifth section models the soil, which can either be infected or healthy. It has not been possible to faithfully capture the model of Webb et al. [9] in PEPA. In particular, the transmission from fleas to prairie dogs, and the reproduction of fleas is driven by density dependent terms that cannot be precisely described in PEPA. The parameters that have been used are shown in Figure 2. Most of them are taken from Webb et al. [9], apart from *fbirthrate* which has been chosen within biologically realistic bounds.

Parameter	Value	Description
birthrate	0.0866/2	Intrinsic rate of increase (host)
K	200	Carrying capacity (host)
totalNbPD	200	Total number of host (including the dead ones)
death_rate_PD	0.0002	Natural mortality rate (host)
transmission_rate	0.09	Flea transmission rate
contact_direct_rate	0.073	Airborne transmission rate
contact_env_rate	0.073/20	Transmission rate from reservoir
incubation_rate	0.21	Incubation period^{-1} (host)
death_rate_inf_PD	0.5	Infected host mortality rate
decay_rate	0.006	Reservoir decay rate
leaveHostRate	0.05	Rate of leaving hosts
a	0.004	Searching efficiency of questing fleas
fdeathrate	0.07	Natural mortality rate (vector)
inf_flea_rate	0.28	Transmission rate: hosts to vector
flea_incubation_period	0.009	Incubation period^{-1} (vector)
finfdeathrate	0.33	Disease-induced mortality rate (vector)
fbirthrate	0.14	Intrinsic rate of increase (vector)

Figure 2: The parameters used in the simulations.

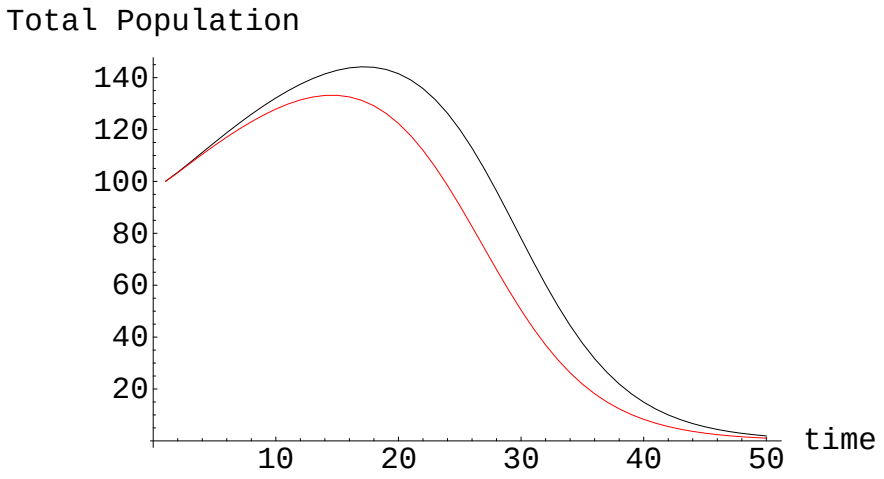


Figure 3: The comparison of the two sets of ODEs: The red line represents the total population of prairie dogs, with equations taken from Webb et al. [9] and the black line is from the equations derived from the PEPA model. Root Mean Square value: 9.19

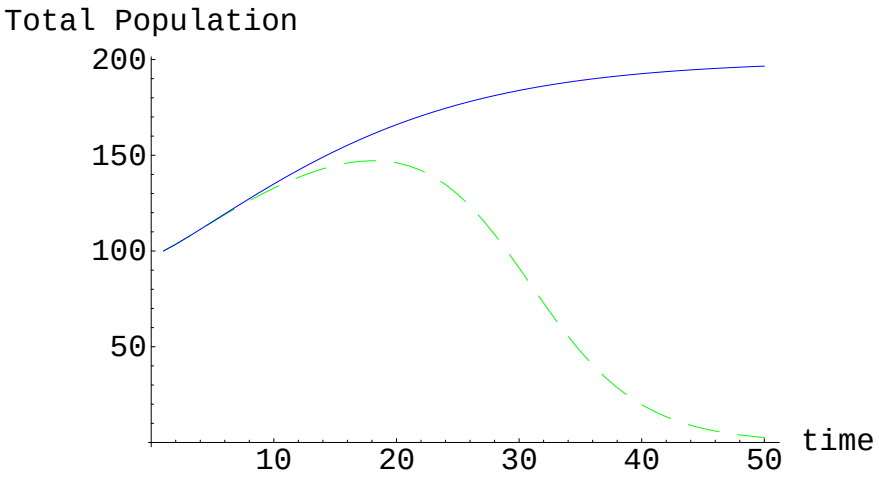


Figure 4: Total population of prairie dogs, when only one vector of transmission is triggered. The blue line represents the number of prairie dogs when only direct transmission is present, and the dashed green one describes what happens when only indirect transmission is considered. The model with fleas only is not represented because it overlaps with the direct transmission model.

3 First results

Ordinary Differential Equations (ODEs) have been derived from the model using the Stirling Amendment algorithm [2], and compared to the set of equations given by Webb et al. [9]. Stochastic simulations appear to produce unreliable results for this model and are therefore not used. For example, the number of sequential components in a particular state can sometimes be calculated to be negative. Clearly, this can never be the case. When the ODEs derived from the PEPA model are used, they match the ODEs from Webb et al. [9], as shown in Figure 3, which is a first step in validating the PEPA model. There is a slight difference at the start, which remains to be understood.

The goal in modelling this system is to consider rigorously the three different transmission methods, and to ascertain if one is more crucial to overall disease dynamics than the others. For this purpose, three submodels are extracted from the model of Figure 1 by setting appropriate parameters to zero. The results, presented in Figure 4 clearly show that the main vector of transmission is indirect transmission. This is a very interesting result, as it contradicts the classical view that fleas are the main vector of transmission. However, a caveat is in order: the parameter driving the indirect transmission is not backed up by data.

4 Conclusion

PEPA has proved to be very capable of describing epidemiological problems. Indeed, although it does not allow directly for some key features present in biological phenomena (e.g. time dependent parameters, density dependent parameters), it provides ways of bypassing the problem, whilst still producting sensible results. However, in order to fully validate this particular PEPA model, a comparison with the data gathered from the field still needs to be carried out. Also, a sensitivity analysis is needed for some of the parameters, in particular the indirect transmission rate, in order to confirm that it is indeed indirect transmission that drives the spread of the disease.

Finally, this first epidemiological model based on a real biological system has provided insights into which issues might arise when modelling biological problems in PEPA. In particular, space and density dependent transmission are two very common features. Therefore, it needs to be determined in which cases it is possible to add one of these features to a PEPA model and how this could be done.

References

[1] C. R. Esleu amd V. H. Haas. Plague in the Western Part of the United States. *U.S Government Printing Office, Washington, DC*, 1940.

[1] A forthcoming paper will give more details about this kind of term.

[2] Soufiene Benkirane, Jane Hillston, Chris McCaig, Rachel Norman, and Carron Shankland. Improved continuous approximation of pepa models through epidemiological examples. *Electron. Notes Theor. Comput. Sci.*, 229(1):59–74, 2009.

[3] M.J. Keeling and C.A. Gilligan. Bubonic plague: a metapopulation model. *P Roy Soc Lond B Bio*, (267):2219–2230, 2000.

[4] W.O. Kermack and A.G. McKendrick. Contributions to the mathematical theory of epidemics. *Proceedings of the Royal Society of London A*, 115:700–721, 1927.

[5] N. B. Kotliar, B. W. Baker, A. D. Whicker, and G. Plump. A critical review of assumptions about the prairie dog as a keystone species. *Environmental Management*, (24):177–192, 1999.

[6] J. D. Poland and A. M. Barnes. *Plague*, volume 1 of *J. H. Steele, ed. CRC handbook series in zoonoses, Section A. Bacterial, rickettsial, and mycotic diseases.* CRC, Boca Raton, FL, 1979.

[7] Paul Stapp, Michael F. Antolin, and Mark Ball. Patterns of extinction in prairie dog metapopulations: plague outbreaks follow el niño events. *Front Ecol Environ*, 5(2):235–240, 2004.

[8] SR Ubico, GO Maupin, KA Fagerstone, and RG McLean. A plague epizootic in the white-tailed prairie dogs (Cynomys leucurus) of Meeteetse, Wyoming. *J Wildl Dis*, 24(3):399–406, 1988.

[9] Colleen T. Webb, Chistopher P. Brooks, Kenneth L. Gage, and Michael F. Antolin. Classic flea-borne transmissoin does not drive plague epizootics in prairie dogs. *PNAS*, 103(16):6236–6241, 2006.

Studying the effects of adding spatiality to a process algebra model

Savi Maharaj, Chris McCaig, Carron Shankland

Department of Computing Science and Mathematics, University of Stirling,
Stirling, Scotland
{sma,cmc,ces}@cs.stir.ac.uk

Abstract

We use NetLogo to create simulations of two models of disease transmission originally expressed in WSCCS. This allows us to introduce spatiality into the models and explore the consequences of having different contact structures among the agents. In previous work, mean field equations were derived from the WSCCS models, giving a description of the aggregate behaviour of the overall population of agents. These results turned out to differ from results obtained by another team using cellular automata models, which differ from process algebra by being inherently spatial. By using NetLogo we are able to explore whether spatiality, and resulting differences in the contact structures in the two kinds of models, are the reason for this different results. Our tentative conclusions, based at this point on informal observations of simulation results, are that space does indeed make a big difference. If space is ignored and individuals are allowed to mix randomly, then the simulations yield results that closely match the mean field equations, and consequently also match the associated global transmission terms (explained below). At the opposite extreme, if individuals can only contact their immediate neighbours, the simulation results are very different from the mean field equations (and also do not match the global transmission terms). These results are not surprising, and are consistent with other cellular automata-based approaches. We found that it was easy and convenient to implement and simulate the WSCCS models within NetLogo, and we recommend this approach to anyone wishing to explore the effects of introducing spatiality into a process algebra model.

1. Introduction

Susceptible-Infected-Recovered (SIR) models are a widely used technique for modelling the spread of infectious disease. These models consist of coupled ordinary differential equations, describing the disease in terms of three population groups: S, susceptible individuals, who have never had the disease; I, infecteds, who can pass on the disease to susceptibles; and R, recovereds who are assumed to be immune from future infection. A key concept in this approach is the "global transmission term" [3] which describes the rate at which susceptibles are converted into infecteds by their contact with previously infected individuals. The correct form of these transmission terms is the subject of much debate. Traditionally, two terms have been used: βSI ("density dependent transmission") and $\beta' SI/N$ ("frequency dependent transmission"). Frequency dependent transmission is typically used for diseases where the number of potentially infectious contacts is considered to be fixed, rather than depending on population density; the classic example is sexually transmitted diseases, if it is assumed that the average person has a fixed number of sexual contacts. Density dependent transmission applies to diseases such as measles, where the number of potentially infectious contacts is higher in a denser population. (We are assuming here that the overall area occupied by the population is fixed, as the transmission terms become more complex if this assumption is relaxed.)

Traditional SIR models describe the situation at a global, aggregate level, and an interesting question has been how to relate this to the behaviour of single individuals. Individual behaviour can be modelled by cellular automata, which can then be analysed by performing simulations, or by a limited set of algebraic techniques such as pair approximation. An alternative technique using process algebra was developed by McCaig [1]. Here, a model of individual behaviour is created using a stochastic process algebra such as WSCCS, and this is then used to derive a set of *mean field equations* representing the emergent population level behaviour.

In this paper we focus on two disease models created in WSCCS, representing density dependent transmission and frequency dependent transmission. These models are presented in full detail in [2] where the techniques of [1] were used to derive mean field equations from these models. The transmission terms can then be derived from these equations, and were shown to be $\beta' SI/N$ (the frequency dependent term) in the model which used frequency-dependent transmission, and βSI (the density dependent term) in the model using density-dependent transmission. These results seem unsurprising, but they appear to be contradicted by results obtained by another team who used a cellular automata approach. Turner et al [4] built cellular

automata models of the two forms of disease transmission and, after running many simulations and using statistical techniques to fit the resulting data to the two transmission terms, concluded that the global frequency dependent term was better at describing both forms of transmission.

Our aim was to investigate the reason for this discrepancy. The WSCCS and cellular automata models differ in a number of details, but the most striking difference is the presence of spatiality in cellular automata. In WSCCS, there is no notion of agents being located in space, and one cannot (or not without some cumbersome encoding) speak of agents being near or far from each other. Contact between agents is therefore entirely random. In cellular automata models, however, agents are usually located on a two-dimensional grid and one can speak of the distance between pairs of agents. In the cellular automata built by [4] contact between agents was determined by the distance between them; in effect, an agent could only make contact with its nearest neighbours. Our suspicion was that this was the reason for the differing results.

We used NetLogo [5] to create implementations of the WSCCS models of both frequency dependent transmission and density dependent transmission. We provided each model with two choices of contact structure between individuals: random mixing, intended to capture the contact structure implicit in WSCCS, and nearest neighbour mixing, which is the contact structure used in the cellular automata of [4]. We then compared the results of simulating these models with the results predicted by the mean field equations derived from the original WSCCS models. In section 2 we explain the WSCCS models, introduce NetLogo, and briefly discuss how the simulations were built. Section 3 describes the results of running the NetLogo simulations and Section 4 discusses our conclusions and possible future work.

2. Using NetLogo to add spatiality to the WSCCS models

The WSCCS models [2] include four types of agents: susceptibles, infecteds, recovereds, and newborns. (Newborns have no interactions with other agents and mature into susceptibles after one time step.) The models consist of three stages. In the first stage all individuals can give birth with a density dependent probability (*p-birth* = *p-birth-0* – $k\ N_t$, where *p-birth-0* is the probability of birth in the absence of crowding, k is a scaling constant, and N_t is the size of the population (which, at this stage, contains no newborns). In addition, the infected individuals can also probabilistically become transmitters. For frequency dependent transmission, this occurs with a fixed probability, *p-contact*. For density dependent transmission, this probability is directly proportional to the size of the population, and is given by the term κN_t, where κ is a scaling constant. This is the only difference between the two kinds of transmission.

In the second stage, each transmitter makes a single contact with another (non-newborn) member of the population. In the final stage, individuals make probabilistic choices, with all probabilities being fixed. Susceptibles which were contacted by a transmitter may either become infected (with probability *p-infect*), die (with probability *p-die*) or remain as susceptible (with probability 1 – *p-infect* – *p-die*). Infecteds either recover (with probability *p-recover*), die (with probability *p-die*), or remain infected (with probability 1 – *p-recover* – *p-die*). Recovereds and uncontacted susceptibles each die (with probability *p-die*) or remain in their current state (with probability 1 – *p-die*). Finally, newborns mature to become susceptible within the next iteration of the model.

The full WSCCS code for each of these models is given in [2], which also gives the derived mean field equations. For frequency dependent transmission, the mean field equations give rise to the transmission term $\beta' S_t I_t / N_t$ (where β' = *p-infect* * *p-contact*), which is the standard frequency dependent transmission term. For density dependent transmission, the derived transmission term is $\beta S_t I_t$ (where β = *p-infect* * κ), which is the standard density dependent transmission term.

Contact between agents in WSCCS is purely random, as there is no notion of agents being spatially located. To study the effect of adding spatiality we translated the WSCCS models into NetLogo. NetLogo [5] is a "cross-platform multi-agent programmable modelling environment" which has been used for building simulations of a very diverse set of phenomena in a wide variety of domains within the physical, natural, and social sciences. It is a descendent of the Logo programming language, which was designed for teaching programming to schoolchildren, and retains some Logo features such as easy and intuitive programmability (as well as the keyword "turtle" to describe an individual agent). However, it also includes many features which make it suitable for serious research. The features most important for our work were the interface builder, which makes it easy to add buttons, sliders, choosers and other I/O features for controlling and monitoring a simulation, and the plotting system, which allows simulation results to be dynamically displayed on screen.

The programming language of NetLogo is well designed to support multi-agent programming. Essentially, mobile agents called "turtles" move over a spatial grid of stationary agents called "patches". (Turtles may also be connected via "link" agents but these were not used.) Turtles may be differentiated into different "breeds", each with their own local variables and methods. In this project, four breeds were defined: `newborns`, `susceptibles`, `infecteds`, and `recovereds`. A large and powerful set of language primitives support interactions between agents. The ease of coding is perhaps best illustrating by showing a sample of code. Figure 1 shows the code for the `transmitters-make-contact` procedure, which

is called at stage 2, after the infected agents have made a probabilistic choice either to become a disease transmitter (`transmitter? = true`) or not. The `ask` command is applied to the agent set `infecteds with [transmitter? = true]`, and causes all such agents to carry out the following block of code. This code first checks whether there remain any other uncontacted non-newborns, as contact will not be possible if all possible contacts have already been made. Contact will then be made according to the chosen contact structure. If random mixing is chosen, then another randomly chosen uncontacted non-newborn will be marked as contacted. If nearest neighbour mixing is chosen, then the contact will be made with a suitable agent located at the minimum distance from the contacting agent. (If there are several agents at the minimum distance, a random choice will be made). The `transmitter?` variable is then reset to false for the next iteration.

```
to transmitters-make-contact
  ask infecteds with [transmitter? = true] [
    if (any? other turtles with [(contacted? = false) and (breed != newborns) ]) [
      if (mixing = "random") [
        ask one-of other turtles with [(contacted? = false) and (breed != newborns) ]
          [set contacted? true]
      ]
      if (mixing = "nearest neighbour") [
        ask min-one-of
           other turtles with [(contacted? = false) and (breed != newborns) ]
          [distance myself]
              [set contacted? true]
      ]
    ]
    set transmitter? false
  ]
end
```

Fig 1: sample NetLogo code

The NetLogo user interface is shown in Fig 2. The "interface" tab has been selected and shows the user interface created for our WSCCS models. (The other two tabs contain documentation and program code.) At the top left there are various buttons, input areas, choosers, and sliders for controlling the simulation and entering parameters. At the bottom left there is a graphical visualization of the simulation. At the right there are three plotting areas, each containing plots of the numbers of infecteds, recovereds, susceptibles, and the total size of the population. These values are plotted against time, measured in "ticks", each corresponding to one iteration of the simulation. The top plot shows the results of the simulation, while the other two plots show the values predicted by the mean field equations corresponding to frequency dependent and density dependent transmission. These plots allow a quick visual comparison between the simulation results and the results predicted by the two transmission terms. For a more rigorous comparison, NetLogo may also be programmed to perform multiple simulation runs over a range of parameter values, and the resulting data may then be subjected to statistical analysis using some other appropriate tool, such as S-PLUS.

3. Results

At this stage in this work we have not yet attempted to perform large-scale multiple simulations or to do any statistical analysis of the simulation results. However, some results already appear to be very likely from observation of several individual simulation runs. It seems safe to say that with random mixing, the simulation results support the conclusions of [2]: density-dependent transmission results in the behaviour described by mean field equations that use the global density-dependent transmission term, and frequency-dependent transmission results in the behaviour described by mean field equations with the global frequency-dependent transmission term. Figure 2 shows an example of a simulation run that illustrates this. In this run, we have selected frequency dependent transmission and random mixing, and the simulation output is clearly a much closer match to the graph of the mean field equations for frequency dependent transmission. (The three sets of graph are separated for clarity, but are drawn to the same scales.)

The results are much less clearcut when nearest neighbour mixing is applied. More work would be needed to study these results systematically, but observations suggest that neither of the two global transmission terms is particularly good at describing either frequency-dependent or density-dependent transmission when nearest-neighbour mixing is used. This is not

very surprising, as there is an assumption of random contact (or "mass action") in the derivation of the global transmission terms, and indeed it appears to be folklore knowledge [7] within the mathematical modelling community that these transmission terms do not generally work for spatially-based models. This is supported by Rhodes and Anderson [6] who developed a cellular automata model involving moving individuals, and varied the "step length", i.e, the distance travelled by an individual at each time step. This approach may be seen as a more general form of our experiment, where random mixing corresponds to the case where an individual may move an arbitrary distance, and nearest-neighbour mixing is the case where the step length is the shortest possible non-zero value. Rhodes and Anderson discovered that the traditional global transmission terms (or "mass-action-based models") are only appropriate if the step length is sufficiently large in comparison with the mean distance between individuals. For shorter step lengths, mass-action is not applicable, and the behaviour of the population is dependent on the value of the step length. Our observations appear to be consistent with these results.

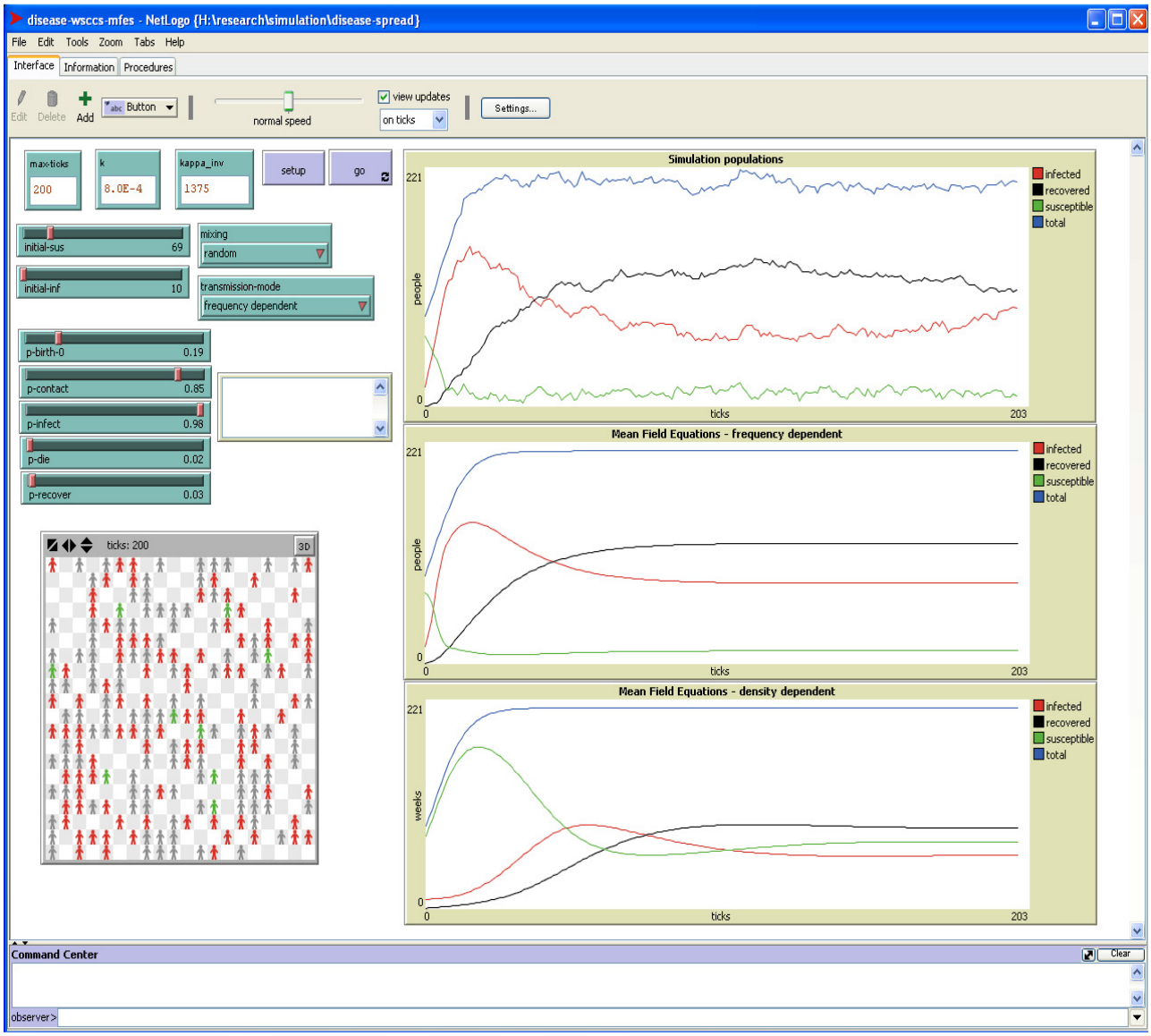


Fig 2: NetLogo interface showing the WSCCS simulation

4. Discussion

We found that NetLogo provided a very effective and easy-to-use method to explore the effect of adding spatiality to a process-algebra based model. The agent-based style of programming used by NetLogo makes it relatively easy to implement process algebra concepts such as agents ("turtles" in NetLogo), different classes of agent ("breeds"), communication and synchronization. There is no direct support for probabilistic actions, however, so these have to be coded explicitly by using random number generation. There are many powerful language constructs supporting spatiality and movement of agents. We also found the support for graphing. We would recommend NetLogo to anyone wishing to investigate the behaviour of a process algebra model when deployed in some particular spatial setting, or with some spatially-described contact structure amongst agents. An example might be exploring the workings of some communications protocol within a specific network configuration. A more ambitious future project would be to develop a systematic method for compiling process algebra models into NetLogo (perhaps with default "random" contact between agents) so as to make it easier to explore the effect of adding spatiality.

References

1. C. McCaig, (2007), From individuals to populations: changing scale in process algebra models of biological systems. PhD thesis, University of Stirling. http://hdl.handle.net/1893/398.
2. C. McCaig, M. Begon, C. Shankland and R. Norman. A rigorous approach to investigating common assumptions about disease transmission, unpublished draft, 2009.
3. M. Begon, M. Bennet, R.G. Bowers, N.P. French, S.M. Hazel, and J. Turner. A clarification of transmission terms in host-microparasite models: numbers, densities and areas. Epidemiol. Infect., 129:147–153, 2002.
4. J. Turner, M. Begon, and R.G. Bowers. Modelling pathogen transmission: the interrelationship between local and global approaches. Proc. Roy. Soc. Lond. B, 270:105–112, 2002.
5. U Wilensky. NetLogo. http://ccl.northwestern.edu/netlogo/ Center for Connected Learning and Computer-Based Modeling Northwestern University, Evanston, IL, 1999
6. C.J. Rhodes and R.M. Anderson. Contact rate calculations for a basic epidemic model. Math. Biosci., 216:56–62, 2008.
7. A Kleczkowski. Personal communication, 2009.

From individual behaviour to population dynamics: changing scale in models of superspreaders

Chris McCaig[1], Mike Begon[2], Carron Shankland[1], and Rachel Norman[1]

[1] Department of Computing Science and Mathematics, University of Stirling, Stirling, FK9 4LA, UK. {cmc,ran,ces}@cs.stir.ac.uk
[2] School of Biological Sciences, University of Liverpool, Liverpool, L69 3BX, UK. {mbegon}@liverpool.ac.uk

1 Introduction

Traditional models of directly transmitted infectious disease [1, 4] split the population into those who are susceptible to the disease, those who are infectious and those who are immune to it. However, it is believed that for a wide range of disease systems there are a few individuals who cause the majority of new infections. These individuals are known as superspreaders [6]. Two mechanisms have been proposed to explain superspreaders and this paper presents our investigation of these proposals.

In the superspreader models that follow the infected portion of the population consists of two distinct groups: standard infected individuals (I) and superspreaders (U). The first mechanism that has been proposed to explain superspreaders is increased infectiousness (also known as supershedders). If a susceptible individual is contacted by a supershedder the probability of becoming infected (p_{iu}) is higher than the probability of becoming infected having been contacted by an infected individual (p_i), i.e. $p_{iu} = \alpha p_i$ for some constant $\alpha > 1$. Supershedders may arise because of a compromised immune system (meaning, for instance, that more of the infectious microorganism is present in their body so the quantity shed is greater) or because of some genetic predisposition that causes them to shed a greater quantity of the infectious microorganism.

In the second proposed mechanism the contact rate for the superspreaders is higher than the contact rate of the infected individuals. We implement this by having the probability that a superspreader will make contact in an iteration of the model (p_{cu}) larger than the probability that and infected individual will make contact (p_c) i.e. $p_{cu} = \alpha p_c$. These superspreaders are more gregarious or well travelled than average and therefore make significantly more contacts than most infected individuals in the population.

The superspreader model presented here was developed in the process algebra Weighted Synchronous Calculus of Communicating Systems (WSCCS) [13], which has proved particularly useful in studying a wide range of biological systems [3, 10–12].

$$
\begin{aligned}
S1 &\stackrel{\text{def}}{=} 1.\surd : S2 \\
I1 &\stackrel{\text{def}}{=} p_{ci}.\surd : I2 \times T2 + (1 - p_{ci}).\surd : I2 \\
U1 &\stackrel{\text{def}}{=} p_{cu}.\surd : U2 \times TU2 + (1 - p_{cu}).\surd : U2 \\
R1 &\stackrel{\text{def}}{=} 1.\surd : R2 \\
\\
S2 &\stackrel{\text{def}}{=} \omega.inf : SI3 + \omega.infU : SU3 + 1.\surd : S3 \\
T2 &\stackrel{\text{def}}{=} \omega.\overline{infect} : 0 + 1.\surd : 0 \\
I2 &\stackrel{\text{def}}{=} \omega.inf : I3 + \omega.infU : I3 + 1.\surd : I3 \\
U2 &\stackrel{\text{def}}{=} \omega.inf : U3 + \omega.infU : U3 + 1.\surd : U3 \\
TU2 &\stackrel{\text{def}}{=} \omega.\overline{infU} : U3 + 1.\surd : U3 \\
R2 &\stackrel{\text{def}}{=} \omega.inf : R3 + \omega.infU : R3 + 1.\surd : R3 \\
B2 &\stackrel{\text{def}}{=} 1.\surd : B3 \\
\\
S3 &\stackrel{\text{def}}{=} 1.\surd : S1 \\
SI3 &\stackrel{\text{def}}{=} (p_i * (1 - p_s)).\surd : I1 + (p_i * p_s).\surd : U1 + (1 - p_i).\surd : S1 \\
SU3 &\stackrel{\text{def}}{=} (p_i u * (1 - p_s)).\surd : I1 + (p_i u * p_s).\surd : U1 + (1 - p_i u).\surd : S1 \\
I3 &\stackrel{\text{def}}{=} p_r.\surd : R1 + (1 - p_r).\surd : I1 \\
U3 &\stackrel{\text{def}}{=} p_r.\surd : R1 + (1 - p_r).\surd : U1 \\
R3 &\stackrel{\text{def}}{=} 1.\surd : R1 \\
\\
Popn &\stackrel{\text{def}}{=} S1\{s\} \times I1\{i\} \times U1\{u\} \times R1\{r\} \lceil \{\surd\}
\end{aligned}
$$

Fig. 1. Superspreader model of disease transmission

2 Model

In this section we give a description of the individual behaviours modelled and also the mean field equations (MFE) that can be derived from the model (making use of a rigorous algorithm described in [9] and [8]). In our the model the population consists of four subpopulations: susceptibles, S; infecteds, I; superspreaders, U; and recovereds, R.

WSCCS is a discrete time process algebra and one consequence of this is that behaviour in our model is separated into several stages. For instance we separate probabilistic behaviours (e.g. birth and death) from those involving contact between individuals (e.g. transmission of infection). By doing this we can more easily reason about the overall behaviour of the system. The different stages happen sequentially during one iteration of the model which is equivalent

to one real time step. The length of the time step depends on the units of the parameters.

We may naively assume that changing the order of these stages would have no effect on the overall average behaviour of the model but in general this is not true. Changing the order in which the stages happen changes the underlying biological assumptions and also alters the resulting MFE [7].

The model in Fig. 1 can be used to represent superspreaders with either increased infectiousness or increased probability of contact by specific choices of parameters. The behaviour in the model in one timestep is separated into three stages. In the first stage infected and superspreader individuals make a probabilistic choice to make an infectious contact in the second stage of the model. Susceptible and recovered individuals do not make any choices at this stage of the model ($S1$ and $R1$ become $S2$ and $R2$ respectively). In addition the infected individuals are able to make an infectious contact in the next stage of the model with probability p_c and superspreaders also choose to make contact in the second stage, with probability p_{cu}.

In the second stage disease transmission occurs. The infecteds and superspreaders that made the probabilistic choice to make contact in the first stage each make a single contact (where contact means the type of contact in which the disease could be transmitted and so will vary in its definition for different diseases). Each type of individual (S, I, U and R) can be contacted once (being exposed to the disease). Only the susceptibles are affected by contact (we assume that recovery from infection confers lifelong immunity) and susceptible individuals that are contacted can contract the disease during the third stage of the model.

In the third stage the susceptible individuals that were contacted probabilistically decide to contract the disease or not. Individuals that were contacted by a standard infected contract the disease with probability p_i, while those contacted by a superspreader become infected with probability p_{iu}. The newly infected individuals themselves become superspreaders with probability p_s. At the same time existing infected and superspreader individuals make the probabilistic choice to recover, with probability p_r.

The mean field equations that describe this generalised model are

$$\begin{aligned}
S_{t+1} &= S_t - \frac{p_i p_{ci} S_t I_t + p_{iu} p_{cu} S_t U_t}{N_t}\,, \\
I_{t+1} &= (1-p_r) I_t + \frac{(1-p_s)(p_i p_{ci} S_t I_t + p_{iu} p_{cu} S_t U_t)}{N_t}\,, \\
U_{t+1} &= (1-p_r) U_t + \frac{p_s (p_i p_{ci} S_t I_t + p_{iu} p_{cu} S_t U_t)}{N_t}\,, \\
R_{t+1} &= R_t + p_r (I_t + U_t), \qquad (1)
\end{aligned}$$

with $N_t = S_t + I_t + U_t + R_t$.

2.1 Contact Superspreaders

The first situation considered features superspreader individuals that are capable of making more contacts than the standard infected individuals. For this case there is no qualitative difference between contact from an infected or superspreader. This is implemented by setting the probability that susceptible individuals become infected after contact the same whether contact is with an I or a U ($p_i = p_{iu}$), and by having the probability that a superspreader will make an infectious contact greater than the probability that an infected individual will make a contact i.e. $p_{cu} = \alpha p_c$ for some $\alpha > 1$.

By applying these parameters to the MFE we find equations to describe the contact superspreaders model:

$$\begin{aligned} S_{t+1} &= S_t - \frac{p_i p_{ci} S_t (I_t + \alpha U_t)}{N_t} , \\ I_{t+1} &= (1 - p_r) I_t + \frac{(1 - p_s) p_i p_{ci} S_t (I_t + \alpha U_t)}{N_t} , \\ U_{t+1} &= (1 - p_r) U_t + \frac{p_s p_i p_{ci} S_t (I_t + \alpha U_t)}{N_t} , \\ R_{t+1} &= R_t + p_r (I_t + U_t), \end{aligned} \tag{2}$$

where $\alpha > 1$ is a constant.

The transmission term in these equations,

$$\frac{p_i S_t (c_i I_t + \alpha c_i U_t)}{N_t} , \tag{3}$$

is of the frequency dependent form [2]. This is because the behaviour described at the individual level is frequency dependent - the probabilities of making contact, p_c and p_{cu}, do not vary with the overall population size. It has already been shown [7] that we can describe behaviour at the individual level that leads to density dependent transmission terms, i.e. $\beta S_t I_t$, in the resulting MFE. Here frequency dependent transmission was used because it is more straightforward to describe in WSCCS and is more reasonable for human diseases.

2.2 Supershedders

The second case we consider features supershedder individuals, with susceptible individuals more likely to become infected after contact with a supershedder than with a standard infected. In this case superspreaders and infecteds are equally likely to make contact ($p_{cu} = p_{ci}$). If a susceptible individual is contacted by a supershedder the probability that it will become infected is greater than the probability that infection will occur after contact with an infected individual ($p_{iu} = \alpha p_i$ for some constant $\alpha > 1$).

From the WSCCS description of this model the mean field equations that we derive are (2), the MFE for the contact superspreader model of Section 2.1. In Section 2.3 we comment on the similarities between the MFE.

2.3 MFE

The two models considered lead to identical systems of mean field equations. This suggests that the actual mechanism by which superspreaders arise is unimportant in determining the population level equations that describe the mean behaviour of the population.

If population level equations were written down directly to model a superspreader system, with frequency dependent transmission and density dependent birth, they would not differ greatly from (2). Kemper [5] proposed an ordinary differential equation model without births or deaths and featuring density dependent transmission in which the transmission terms were: $(r_1 I + r_2 U)S$ - the rate at which susceptibles contract the disease; $\beta(r_1 I + r_2 U)S$ - the rate at which susceptibles become standard infected individuals; $(1-\beta)(r_1 I + r_2 U)S$ - the rate at which susceptibles become superspreaders. Kemper's model did not explicitly consider which of the two proposed mechanisms (increased contacts or increased infectiousness) give rise to superspreaders. Instead it merely captures the idea that a small proportion of the infected individuals are responsible for the majority of new infections.

The difference with our MFE, however, is that we can be sure that our MFE are a direct consequence of the individual level assumptions made, having been rigorously derived from the individual based description of the model. This allows us additional ways to study the system, for example by performing stochastic simulations of the model allowing us to study the variability in the system as well as the mean behaviour considered by the MFE.

It is worth noting that if all we are interested in is the mean behaviour then it is possible to choose different parameter values that would give the same overall mean behaviour from a model without superspreaders. For instance considering a supershedder system (with $p_{iu} = \alpha p_i$), p_s will be the mean proportion of infected individuals that are superspreaders and it would be possible to design a model without supershedders with a different probability of infection p_i' that would have the same mean behaviour. This is done for our example by setting $\alpha = 1$, so that the supershedders have the same behaviour as the standard infecteds, and by setting $p_i = p_{iu} = (1-p_s)p_i + \alpha p_i p_s$ - all other parameter values are unchanged.

3 Conclusions

Two different mechanisms have been proposed by which superspreaders may occur in populations [5, 6]. By developing individual based models, and rigorously deriving population level equations to describe the mean behaviour, we have demonstrated that changing the mechanism at the individual level has little effect on the population as a whole. In addition we have demonstrated that, by choosing different parameter values, it is possible to write a model without superspreaders in which the average behaviour will be identical to the model with superspreaders.

References

1. R.M. Anderson and R.M. May. *Infectious diseases of humans : dynamics and control*. Oxford University Press, 1991.
2. M. Begon, M. Bennet, R.G. Bowers, N.P. French, S.M. Hazel, and J. Turner. A clarification of transmission terms in host-microparasite models: numbers, densities and areas. *Epidemiology and infection*, 129:147–153, 2002.
3. M. J. Hatcher and C. Tofts. The evolution of polygenic sex determination with potential for environmental manipulation. Technical Report UMCS-95-4-2, Department of Computer Science, University of Manchester, 1995.
4. H.W. Hethcote. The mathematics of infectious diseases. *SIAM Review*, 42:599–653, 2000.
5. J.T. Kemper. Identification of superspreaders for infectious-disease. 48:111–127, 1980.
6. J.O. Lloyd-Smith, S.J. Schreiber, P.E. Kopp, and W.M. Getz. Superspreading and the effect of individual variation on disease emergence. *Nature*, 438:355–359, 2005.
7. C. McCaig. *From individuals to populations: changing scale in process algebra models of biological systems*. PhD thesis, University of Stirling, 2007. http://hdl.handle.net/1893/398.
8. C. McCaig, R. Norman, and C. Shankland. Deriving mean field equations from large process algebra models. Technical Report CSM-175, Department of Computing Science and Mathematics, University of Stirling, March 2008. http://www.cs.stir.ac.uk/research/publications/techreps/pdf/TR175.pdf.
9. C. McCaig, R. Norman, and C. Shankland. Process algebra models of population dynamics. In *Algebraic Biology*, volume 5147 of *Lecture Notes in Computer Science*, pages 139–155. Springer-Verlag, 2008.
10. R. Norman and C. Shankland. Developing the use of process algebra in the derivation and analysis of mathematical models of infectious disease. In *Computer Aided Systems Theory - EUROCAST 2003*, volume 2809 of *Lecture Notes in Computer Science*, pages 404–414. Springer-Verlag, 2003.
11. D. Sumpter. *From Bee to Society: an agent based investigation of honeybee colonies*. PhD thesis, UMIST, 2000.
12. C. Tofts. Using process algebra to describe social insect behaviour. *Transactions of the Society for Computer Simulation*, 9:227–283, 1993.
13. C. Tofts. Processes with probabilities, priority and time. *Formal Aspects of Computing*, 6:536–564, 1994.