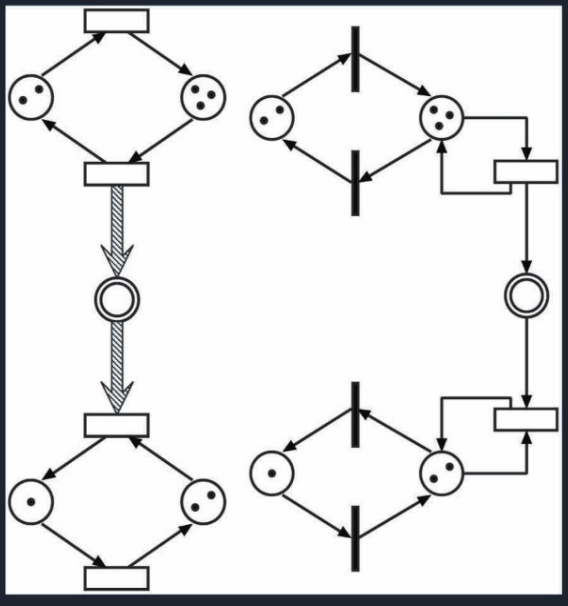
Marco Bernardo
Jane Hillston (Eds.)

Tutorial

LNCS 4486

Formal Methods for Performance Evaluation

7th International School on Formal Methods for the Design of Computer, Communication and Software Systems, SFM 2007 Bertinoro, Italy, May/June 2007, Advanced Lectures



Springer

Lecture Notes in Computer Science 4486

Commenced Publication in 1973
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Editorial Board

Marco Bernardo Jane Hillston (Eds.)

Formal Methods for Performance Evaluation

7th International School on Formal Methods for the Design of Computer, Communication and Software Systems, SFM 2007
Bertinoro, Italy, May 28-June 2, 2007
Advanced Lectures

Springer

Volume Editors

Marco Bernardo
Università di Urbino "Carlo Bo"
Istituto di Scienze e Tecnologie dell'Informazione
Piazza della Repubblica 13, 61029 Urbino, Italy
E-mail: bernardo@sti.uniurb.it

Jane Hillston
The University of Edinburgh
Laboratory for Foundations of Computer Science
Mayfield Road, Edinburgh EH9 3JZ, UK
E-mail: jeh@inf.ed.ac.uk

Library of Congress Control Number: 2007926022

CR Subject Classification (1998): D.2.4, D.2, D.3, F.3, C.3, C.2.4

LNCS Sublibrary: SL 2 – Programming and Software Engineering

ISSN 0302-9743
ISBN-10 3-540-72482-6 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-72482-7 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12063000 06/3180 5 4 3 2 1 0

Preface

This volume presents the set of papers accompanying the lectures of the seventh International School on Formal Methods for the Design of Computer, Communication and Software Systems (SFM).

This series of schools addresses the use of formal methods in computer science as a prominent approach to the rigorous design of computer, communication and software systems. The main aim of the SFM series is to offer a good spectrum of current research in foundations as well as applications of formal methods, which can be of help for graduate students and young researchers who intend to approach the field.

SFM 2007 was devoted to formal techniques for performance evaluation and covered several aspects of the field, including formalisms for performance modeling (Markov chains, queueing networks, stochastic Petri nets, and stochastic process algebras), equivalence checking and model checking, efficient solution techniques, and software performance engineering.

The opening paper by Stewart presents Markov chains, the fundamental performance modeling formalism in use since the early 1900s. The author outlines the events that have led to the present state of the art in the numerical approach to Markov chain performance modeling and describes current solution methods and ongoing research efforts.

The paper by Balsamo and Marin is about queueing networks, a class of stochastic models extensively applied to represent and analyze resource-sharing systems such as communication and computer systems. The authors mostly focus on product-form queueing networks, which allow one to define efficient algorithms to evaluate average performance measures.

The paper by Balbo illustrates generalized stochastic Petri nets, a modeling formalism that can be conveniently used both for the functional verification of complex models of discrete-event dynamic systems and for their performance and reliability evaluation.

The paper by Clark, Gilmore, Hillston, and Tribastone provides an introduction to stochastic process algebras and their use in performance modeling, with a focus on the PEPA formalism. The authors describe the compositional modeling capabilities of the formalism and the tools available to support Markov-chain-based analysis.

The paper by Bernardo defines and compares several Markovian behavioral equivalences with respect to a number of criteria such as their discriminating power, the exactness of the Markov-chain-level aggregations they induce, the achievement of the congruence property, the existence of sound and complete axiomatizations, the existence of logical characterizations, and the existence of efficient verification algorithms.

The paper by Kwiatkowska, Norman, and Parker presents an overview of model checking for both discrete-time and continuous-time Markov chains, which deals with algorithms for verifying them against specifications written in probabilistic extensions of temporal logic, including quantitative properties with rewards. The authors also outline the main features supported by the probabilistic model checker PRISM.

The paper by Gribaudo and Telek summarizes the basic concepts and the potential use of Markov fluid models, together with the factors that determine the limits of their solvability and practical guidelines that can be extracted from these factors to establish the applicability of fluid models in practice.

The paper by Knottenbelt and Bradley explores an array of techniques for analyzing stochastic performance models with large state spaces. The authors concentrate on explicit techniques suitable for unstructured state spaces and show how memory and run-time requirements can be reduced using a combination of probabilistic algorithms, disk-based solution techniques, and communication-efficient parallelism based on hypergraph partitioning.

The paper by Ciardo discusses some important classes of decision diagrams and shows how they can be effectively employed to derive symbolic algorithms for the analysis of large discrete-state models. In particular, the author presents both explicit and symbolic algorithms for state-space generation, CTL model checking, and continuous-time Markov chain solution.

The paper by Smith reviews the origins of software performance engineering (SPE) and covers its fundamental elements: the data required, the software performance models, and the SPE process. The author also illustrates how to apply the modeling and analysis techniques and reports on the current status as well as the outstanding problems.

The closing paper by Woodside is about using the SPT/MARTE annotations to capture important performance features of a software design, such as platform operations, component submodel composition, state machine uses, and communication costs and delays. The author also addresses the relationship of the annotated design model to the different kinds of performance model that can be extracted.

We believe that this book offers a comprehensive view of what has been done and what is going on worldwide in the field of formal methods for performance evaluation. We wish to thank all the lecturers and all the participants for a lively and fruitful school. We also wish to thank the entire staff of the University Residential Center of Bertinoro for the organizational and administrative support. Finally, we are very grateful to BiCi – Bertinoro international Center for informatics, which kindly provided a sponsorship for this event under the Leonardo Melandri Program.

June 2007

Marco Bernardo
Jane Hillston

Table of Contents

Performance Modelling and Markov Chains 1
William J. Stewart

Queueing Networks 34
Simonetta Balsamo and Andrea Marin

Introduction to Generalized Stochastic Petri Nets 83
Gianfranco Balbo

Stochastic Process Algebras 132
Allan Clark, Stephen Gilmore, Jane Hillston, and Mirco Tribastone

A Survey of Markovian Behavioral Equivalences 180
Marco Bernardo

Stochastic Model Checking 220
Marta Kwiatkowska, Gethin Norman, and David Parker

Fluid Models in Performance Analysis 271
Marco Gribaudo and Miklós Telek

Tackling Large State Spaces in Performance Modelling 318
William J. Knottenbelt and Jeremy T. Bradley

Data Representation and Efficient Solution: A Decision Diagram Approach 371
Gianfranco Ciardo

Introduction to Software Performance Engineering: Origins and Outstanding Problems 395
Connie U. Smith

From Annotated Software Designs (UML SPT/MARTE) to Model Formalisms 429
Murray Woodside

Author Index 469

Performance Modelling and Markov Chains

William J. Stewart*

Department of Computer Science, North Carolina State University,
Raleigh, NC 27695, USA
billy@csc.ncsu.edu

Abstract. Markov chains have become an accepted technique for modeling a great variety of situations. They have been in use since the early 1900's, but it is only in recent years with the advent of high speed computers and cheap memory that they have begun to be applied to large-scale modeling projects. This paper outlines the events that have lead to the present state-of-the-art in the numerical approach to Markov chain performance modeling and describes current solution methods and ongoing research efforts.

1 Introduction

1.1 A.A. Markov

It is appropriate in a talk of this nature to include a few sentences about the life and work of A.A. Markov. Three sources were used to gather the information in this section, namely a paper by Basherin and Naoumov, presented at the Fourth International Conference on the Numerical Solution of Markov Chains, [2], a paper by Seneta [67] given at the 2006 Markov Anniversary Meeting, and a web page maintained by the School of Mathematics and Statistics at the University of St Andrews, Scotland [74]. Andrei Andreevich Markov was born on June 14, 1856, in Ryazan, Russia. He was the fourth of his father's 6 children by his first marriage. His father also had 3 children by his second wife. An inborn deformity of the knee meant that the very young Markov had to use crutches. At age ten he was operated on and thereafter walked with a slight limp. However, it was leg problems that lead to his death. In later life, he developed an anuerysm in the leg which required multiple surgeries, one of which proved fatal. He died on July 20, 1922 at the age of 66 and is buried in the Mytrophany Cemetery in St. Petersburg.

Four years after Markov's birth, the family moved to St. Petersburg where Markov attended school. Apparently he was unsuccessful at many subjects, except mathematics, at which he excelled. He attended the University of St. Petersburg where he studied under P.L. Chebyshev. He was awarded the gold medal and was offered an academic position within the university. His doctorate (1884) was entitled "On Certain Applications of the Algebraic Continuous Fractions". When Chebychev left the university in 1883, Markov took over his probability theory course. Twenty years later he was made an honorary professor, and shortly thereafter, he retired, although he continued to lecture on probability theory and the theory of continuous fractions.

* Corresponding author.

M. Bernardo and J. Hillston (Eds.): SFM 2007, LNCS 4486, pp. 1–33, 2007.

A.A. Markov married Maria Valvatieva, the daughter of the owner of the estate on which Markov's father worked, in 1883. They has a son who was given the same name, Andrei Andreevich, and who became a renown mathematician in his own right. A.A. Markov Jr. worked in the fields of algebra, topology and mathematical logic and headed the Department of Mathematical Logic at Moscow State University. Interestingly, this position was filled by A.N. Kolmogorov after Markov Jr.'s death. It was Kolmogorov who laid the foundations for the general theory of Markov processes.

Much of Markov's work was concerned with investigations into the weak law of large numbers and the central limit theorem. His introduction of the Markov chain was to show that Chebyshev's approach to extending the weak law of large numbers to sums of dependent random variables could be extended even further. He introduced a *simple chain* as "an infinite sequence $x_1,\ x_2,\ \ldots,\ x_k,\ x_{k+1},\ \ldots$ of variables connected in such a way that x_{k+1} for any k is independent of $x_1,\ x_2,\ \ldots,\ x_{k-1}$, in case x_k is known". Furthermore he extended this to *complex chains* in which "every number is directly connected not with a single but with several preceding numbers". For the most part, his studies on chains involved simple homogeneous chains. His works address the concept of irreducibility and he shows that the dominant eigenvalue of an irreducible Markov chain must be one and that no other eigenvalue can exceed this in modulus. Markov's work has been the basis for much research and provides a powerful method of analysis that is in vogue today.

1.2 Vic Wallace and the Recursive Queue Analyzer

To the best of this author's knowledge, the first applications of Markov chains to the performance evaluation of computer systems occurred in the early 1960's at the University of Michigan. The *Michigan Computer Modelling Project* was established in the Electrical Engineering department under the direction of Dr. Harrry Goode, the father of "Systems Engineering". Unfortunately Goode was killed in a car accident shortly after its establishment. The project was supported by the *Rome Air Development Center* and later by ARPA. The participants in the original project included Vic Wallace, as principal investigator, Richard Evans, Eugene Lawler, Dennis Fife, Robert Carlson, Richard Rosenberg, Robert Rosen and John Smith. The original project was followed by others under the direction of Keki Irani. The specification for the first software package for Markov modelling, the *Recursive Queue Analyzer*, RQA-0, appeared in the first report of the research group in 1964. This was a prototype program for the numerical solution of the equilibrium state probabilities of Markov chain models and included the use of sparse matrix algorithms and compact storage for the transition matrix. A software package based on these specifications, called RQA-1, was written by Wallace and Rosenberg and appeared in 1966.

In a paper presented at the first international conference on the numerical solution of Markov chains [78], (and from which much of the content of this section is extracted), Wallace cites a number of reasons why numerical solution techniques took so long to develop and suggests that these same reasons were still present some 25 years after the introduction of the Recursive Queue Analyzer. Among these was the idea that computer Markov modelling was too new and unfamiliar, being more "mathematical" and abstract, and not to be trusted. He also suggested that other techniques like product form

networks and simulation, were distracting attention because they could be more readily understood, and there were still enough problems that they could solve. Perhaps more significantly, he postulated that the numerical techniques tended to be quite fragile, that traps abounded to snare the unwary!

To illustrate one aspect of this, Wallace refers to an unsuccessful experience in the application of RQA-1. He recalls in the above referenced paper, that although they had been doing numerical Markov analysis for many years, "I can recall the embarrassment in 1970 of encountering a seriously ill-conditioned model for the first time while confidently engaged in consulting. No adjustments seemed to eliminate the problem, and tens of thousands of iterations were getting nowhere." It turns out that the corporation for which Wallace was consulting was ICL, International Computers Limited, England. The problem was to model a SCAN strategy on a disk store with a large number of cylinders. The difficulty, it later turned out, was the occurrence of multiple eigenvalues close to unity. The problems that Wallace faced are still with us today, even though much progress has been made in the intervening years. These are the problems of finding stable and efficient algorithms for computing numerical solutions and data structures in which to store and manipulate the transition matrices.

1.3 Alan Scherr

Alan Scherr enters the picture as a PhD student at M.I.T. in the process of completing his thesis in electrical engineering in 1965. The university had recently acquired a new "Compatible Time Sharing System" which allowed 300 users to simultaneously access its software and run their programs. Scherr's performance problem was to characterize the system's usage which he did by extensive simulations. On presenting his thesis he was told that it just was not "academic" enough, that it needed more mathematical formulas. To jump this final hurdle, Scherr used some techniques he had learned in a recently completed operations research course to construct a very primitive Markov chain model. He designated the integer n to represent the state of the entire system when n users had submitted their requests — which leaves $300 - n$ users busy typing and preparing to submit their request. What was truly amazing and unexpected was the accuracy with which this simple model appeared to capture the actual behavior of the computer. It was the success of this simple model that encouraged others to adopt the Markov chain approach for performance evaluation. Scherr's thesis was later awarded the Grace Murray Hopper award by the ACM. More information on the work of Scherr is can be found in a paper by Von Hilgers and Langville [76].

2 Context for Current State-of-the-Art

In the context of Performance Evaluation, numerical analysis methods refer to those methods which work with a Markov chain representation of the system under evaluation and use techniques from the domain of numerical analysis to compute stationary and/or transient state probabilities or other measures of interest. It is often possible to represent the behavior of a physical system by describing all the different states that it can occupy and by indicating how the system moves from one state to another in time.

If the time spent in any state is exponentially distributed, the system may be represented by a *Markov process*. Even when the system does not possess this exponential property explicitly, it is usually possible to construct a corresponding implicit representation. When the state space is discrete, the term *Markov chain* is employed. The system being modelled by the chain is assumed to occupy one and only one of these states at any moment in time and the evolution of the system is represented by transitions of the Markov chain from one state to another. The information that is most often sought from such a model is the probability of being in a given state or subset of states at a certain time after the system becomes operational. Often this time is taken to be sufficiently long that all influence of the initial starting state has been erased. The probabilities thus obtained are referred to as the *long-run* or *stationary probabilities*. Probabilities at a particular time t are called *transient probabilities*.

It follows that the three steps involved in carrying out this type of evaluation are firstly, to describe the system to be analyzed as a Markov chain; secondly, to determine from this Markov chain description, a matrix of transition rates or probabilities; and thirdly, from this matrix representation, to numerically compute all performance measures of interest. The first involves characterizing the states of the system and formulating the manner in which it moves from one state to another; the second requires finding a manner in which to store the transition matrix efficiently; the third requires the application of matrix equation solving techniques to compute stationary or transient probabilities.

The Recursive Queue Analyzer, RQA-1, [77], essentially avoided the first step by requiring a user to describe the transition matrix directly. Since the envisaged applications derived from queueing networks, the nonzero elements in the transition matrix often repeat at well defined patterns in the matrix. RQA-1 defined a data structure which attempted to capture such regularities and to store them as the trio (nonzero element, pattern, initialization point). The amount of storage used was therefore minimized. The numerical solution technique employed by RQA-1 was the *Power method*. In the literature, the authors reported some success with this approach, [78].

This was followed in the early 1970's by this author's *Markov Chain Analyzer*, MARCA, [73]. MARCA provided a means of expressing a Markov chain as a system of "Balls and Buckets", essentially allowing a single state of the chain to be represented as a vector, the *state descriptor vector*. The user has to characterize the way in which the system changes states by describing the interactions among the components of the state descriptor vector. With this information MARCA automatically generates the transition rate matrix and stores it in a compact form. The original solution method used in MARCA was simultaneous iteration, a forerunner of the currently popular projection methods. In 1974, a restriction of MARCA to queueing networks was developed and incorporated into QNAP (Queueing Network Analysis Package), [56].

The 80's witnessed the popularization of the matrix-geometric approach of Neuts, [49], and the establishment of Generalized Stochastic Petri Nets (GSPN) as a valuable modelling paradigm, [1, 15]. These advances were accompanied by a flurry of activity in numerical aggregation and disaggregation methods, [13, 44, 65] — extending the seminal work of Courtois, [18], on nearly completely decomposable systems. This period also saw advances in the computation of bounds, [26, 66], in specification

techniques, [6], in state-space exploration, [12, 7], and so on. The most popular applications were, and still are, in the fields of computer communications and reliability modelling, [36, 40, 81, 82]. And, of course, this period was also rich in advances led by the numerical analysis community, especially in the development of projection methods, preconditioning techniques and in sparse matrix technology in general, [62].

To this day, research continues along the same paths. Advances continue to be made in all the aforementioned areas. In addition we see an increased emphasis placed on stochastic automata networks (SANs), and other structured analysis approaches, [10, 11, 16, 41, 53, 72]. These have advanced hand in hand with compositional approaches, such as the stochastic process algebra package, PEPA, [32]. Given the ease of with which GUIs (Graphical User Interaces) can now be programmed and the availability of cheap memory and fast CPUs, many more numerical analysis software packages specificially designed for performance evaluation have made their apparation. This period also witnessed international conferences devoted to the topic, the first in 1990 and the second in 1995. It is significant that the third and fourth in this series were held jointly — with the PNPM (Petri Nets and Performance Models) conference and the PAPM (Process Algebra and Performance Modelling) conference in 1999 in Zaragoza, Spain and with the PNPM (Petri Nets and Performance Models) conference and the Performance Tools and Techniques, Tools'03 conference in Urbana-Champaign in 2003. In a short paper like this, it is not possible to cover all aspects of the current state-of-the-art in anything other than a perfunctory fashion. To offset this to some degree, an extensive bibliography is provided.

3 The Numerical Solution of Markov Chains

In Markov chain performance evaluation, all the desired performance measures are largely computed from the stationary and transient distributions of the Markov chain. In particular, the computation of stationary distributions is generally referred to as solving the global balance equations. These performance measures are computed from the stochastic transition probability matrix P of the Markov chain The elements p_{ij} of this matrix P are the *conditional probabilities* that on leaving state i the Markov chain next moves to state j. The relevant equations may be written as

$$\pi P = \pi, \tag{1}$$

or alternatively, as

$$\pi Q = 0, \tag{2}$$

where $P = Q\Delta t + I$ and $\Delta t \leq (\max_i |q_{ii}|)^{-1}$. When we perform this operation we essentially convert the continuous-time system represented by the *transition rate* matrix, Q, to a discrete-time system represented by the stochastic *transition probability* matrix, P. In the discrete-time system, transitions take place at intervals of time Δt, this parameter being chosen so that the probability of two transitions taking place in time Δt is negligible. The stationary distribution, π, may be computed from either of these equations. The transient distribution is computed form the Chapmann-Kolmogoroff differential difference equations

$$\begin{cases} \dfrac{d\pi(t)}{dt} = \pi(t)Q,\ t \in [0,\ T] \\ \pi(0) \quad = \pi_0 \qquad \text{an initial probability distribution.} \end{cases}$$

We shall discuss computational aspects of the stationary and transient distributions momentarily, but for the moment we shall address a topic that is receiving much attention in the current literature, the set of **right-hand** eigenvectors of P. An understanding of the significance of these vectors is becoming increasing important in applications such as data mining and search engine development.

3.1 Significance of Subdominant, Right-Hand Eigenvectors

It is known that a *left-hand* eigenvector corresponding to a unit eigenvalue of the stochastic transition probability matrix of a Markov chain is its stationary probability vector. As yet, no physical significance has been ascribed to the left-hand eigenvectors corresponding to eigenvalues different from unity. The situation is otherwise for the set of right-hand eigenvectors. When the Markov chain under consideration is irreducible and noncyclic, its stochastic matrix has a single eigenvalue of modulus 1; i.e., the unit eigenvalue. In this case some information concerning the tendencies of the states to form groups may be obtained from an examination of the right-hand eigenvectors corresponding to the *subdominant* eigenvalues, i.e., the eigenvalues with modulus closest to but strictly less than 1.0. The reason is as follows:

The equilibrium position of the system is defined by the stationary probability vector, i.e., the left-hand eigenvector corresponding to the unit eigenvalue. With each state of the system can be associated a real number, which determines its "distance" from this equilibrium position. This distance may be regarded as the number of iterations (or the length of time) required to reach the equilibrium position if the system starts in the state for which the distance is being measured. Such measurements are, of course, only relative, but they serve as a means of comparison among the states.

Let the row vector $w_i^{(1)} = (0, 0, \ldots, 1, \ldots, 0)$ with i^{th} component equal to 1, denote that initially the system is in state i. We shall assume that P possesses a full set of n linearly independent eigenvectors. Similar results may be obtained when eigenvectors and principal vectors are used instead. Let $x_1, x_2, \ldots, x_n$ be the left-hand eigenvectors of P (i.e., $x_j^T P = \lambda_j x_j^T$ for all $j = 1, 2, \ldots, n$), arranged into descending order according to the magnitude of their corresponding eigenvalues. Writing $w_i^{(1)}$ as a linear combination of these eigenvectors, we have

$$w_i^{(1)} = c_{i1}x_1^T + c_{i2}x_2^T + \ldots + c_{in}x_n^T$$

where $c_{i1}, c_{i2}, \ldots, c_{in}$ are the constants that define the linear combination. Repeated postmultiplication of $w_i^{(1)}$ by P yields the steady-state probability vector. We have

$$\begin{aligned} w_i^{(1)}P &= c_{i1}x_1^T P + c_{i2}x_2^T P + \ldots + c_{in}x_n^T P \qquad (3) \\ &= c_{i1}x_1^T + c_{i2}\lambda_2 x_2^T + \ldots + c_{in}\lambda_n x_n^T = w_i^{(2)}, \qquad (4) \end{aligned}$$

and in general

$$w_i^{(k+1)} = c_{i1}x_1^T + c_{i2}\lambda_2^k x_2^T + \ldots + c_{in}\lambda_n^k x_n^T.$$

If the system initially starts in some other state $j \neq i$, we have

$$w_j^{(k+1)} = c_{j1}x_1^T + c_{j2}\lambda_2^k x_2^T + \ldots + c_{jn}\lambda_n^k x_n^T.$$

Since only the constant coefficients differ, the difference in the length of time taken to reach the steady state from any two states i and j depends only on these constant terms. Further, if λ_2 is of strictly larger modulus than λ_3, $\lambda_4, \ldots$, then for large k, $\lambda_2^k \gg \lambda_l^k$ for $l \geq 3$, and it is the terms c_{i2} and c_{j2} in particular that contribute to the difference. Considering all possible starting states, we obtain

$$\begin{pmatrix} w_1^{(k+1)} \\ w_2^{(k+1)} \\ \vdots \\ w_n^{(k+1)} \end{pmatrix} = \begin{pmatrix} c_{11}x_1^T + c_{12}\lambda_2^k x_2^T + \ldots + c_{1n}\lambda_n^k x_n^T \\ c_{21}x_1^T + c_{22}\lambda_2^k x_2^T + \ldots + c_{2n}\lambda_n^k x_n^T \\ \vdots \\ c_{n1}x_1^T + c_{n2}\lambda_2^k x_2^T + \ldots + c_{nn}\lambda_n^k x_n^T \end{pmatrix},$$

i.e.,

$$W^{(k+1)} = \begin{pmatrix} c_{11} & c_{12} & \ldots & c_{1n} \\ c_{21} & c_{22} & \ldots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \ldots & c_{nn} \end{pmatrix} \begin{pmatrix} 1 & & & \\ & \lambda_2^k & & \\ & & \ddots & \\ & & & \lambda_n^k \end{pmatrix} \begin{pmatrix} x_1^T \\ x_2^T \\ \vdots \\ x_n^T \end{pmatrix} \equiv C\Lambda^k X^T.$$

To obtain the matrix C, consider the following: the matrix $W^{(1)}=(w_1^{(1)}, w_2^{(1)}, \ldots, w_n^{(1)})^T$ was originally written in terms of the set of left-hand eigenvectors as

$$W^{(1)} = CX^T,$$

but since $W^{(1)} = I$, we obtain

$$I = CX^T,$$

i.e., $C = (X^T)^{-1} = Y$, the set of right-hand eigenvectors of P. Therefore, it is from the second column of the matrix C, i.e., the subdominant right-hand eigenvector of the matrix P, that an appropriate measure of the relative distance of each state from the stationary probability vector may be obtained. The third and subsequent columns may be employed to obtain subsidiary effects.

States whose corresponding component value in this vector is large in magnitude are, in a relative sense, far from the equilibrium position. Also, the states corresponding to component values that are relatively close together form a cluster, or a subset, of states. If, for example, the components of this vector are close either to $+1$ or to -1, then it may be said that those states corresponding to values close to $+1$ form a subset of states that is far from the remaining states, and vice versa. In this manner, it may be possible to determine which states constitute near essential and near cyclic subsets of states. This technique has been proven useful in a variety of applications in which groups of like states need to be identified and isolated.

3.2 Steady State Distributions

We now turn to the computation of steady-state distributions. A number of powerful numerical procedures are available to us, but unfortunately there is not a single one that works well in all circumstances. It is necessary to choose a particular method that responds well to the size, structure and required performance measures. We examine the various possibilities in the next few sections.

3.2.1 Direct Methods

Direct methods, in contrast to iterative and projection methods, perform a fixed number of numerical operations to compute a solution to a system of equations. All direct methods for systems of linear equations are based on Gaussian elimination. In our case, we apply these methods to equation (1) which is a homogeneous system of linear equations. If the Markov chain is ergodic, the fact that the system of equations is homogeneous does not create any problems, since we may replace any of the n equations by the normalizing equation, $\sum_{j=1}^{n} \pi_j = 1$, and thereby convert it into a nonhomogeneous system with nonsingular coefficient matrix and nonzero right hand side. The solution in this case is well defined. It turns out that replacing an equation with the normalizing equation is not really necessary. The usual approach taken is to construct an LU decomposition of Q and replace the final zero diagonal element of U with an arbitrary value. The solution computed by backsubstitution on U must then be normalized. Furthermore, since the diagonal elements are equal to the negated sum of the off-diagonal elements (Q is, in a restricted sense, diagonally dominant), it is not necessary to perform pivoting while computing the LU decomposition. This simplifies the algorithm considerably. The problem of the size and nonzero structure (the placement of the nonzero elements within the matrix) still remain. Obviously this method will work, and work well, when the number of states is small. It will also work well when the nonzero structure of Q fits into a narrow band along the diagonal. In these cases a very stable variant, referred to as the GTH (Grassmann, Taskar and Heyman, [30]) algorithm may be used. In this variant, all subtraction is avoided by computing diagonal elements as the sum of off-diagonal elements. This is possible since the zero-row-sum property of an infinitesimal generator is invariant under the basic operation of Gaussian elimination, namely adding a multiple of one row into another. For an efficient implementation, the GTH variant requires convenient access to both the rows and the columns of the matrix.

3.2.2 Basic Iterative Methods

When the number of states becomes large and the structure in not banded, the direct approach loses its appeal and one is obliged to turn to other methods. For iterative methods we first take the approach of solving equation (2) in which P is a matrix of transitions probabilities. Let the initial probability distribution vector be given by $\pi^{(0)}$. After the first transition, the probability vector is given by $\pi^{(1)} = \pi^{(0)}P$; after k transitions it is given by $\pi^{(k)} = \pi^{(k-1)}P = \pi^{(0)}P^k$. If the Markov chain is ergodic, then $\lim_{k\to\infty} \pi^{(k)} = \pi$. This method of determining the stationary probability vector, by successively multiplying some initial probability distribution vector by the matrix of transition probabilities, is called the *Power* method. Observe that all that is required is a vector-matrix multiplication operation. This may be conveniently performed on

sparse matrices that are stored in compact form. Because of its simplicity, this method is widely used, even though it often takes a very long time to converge. Its rate of convergence is a function of how close the subdominant eigenvalue of P is to its dominant unit eigenvalue. In models in which there are large differences in the magnitudes of transition rates, the subdominant eigenvalue can be pathologically close to one, so that to all intents and purposes, the Power method fails to converge.

It is also possible to apply iterative equation solving techniques to the system of equations (1). The well-known *Jacobi* method is closely related to the Power method, and it also frequently takes very long to converge. A better iterative method is that of *Gauss-Seidel*. Unlike the previous two methods, in which the equations are only updated after each completed iteration, the Gauss-Seidel method uses the most recently computed values of the variables as soon as they become available and, as a result, almost always converges faster than Jacobi or the Power method. All three methods can be written so that the only numerical operation is that of forming the product of a sparse matrix and a probability vector so all are equal from a computation per iteration point of view.

3.2.3 Block Methods

If the state space of the Markov chain can be meaningfully partitioned into N subsets of size $n_1, n_2, \ldots, n_N$ with $\sum_{i=1}^{N} = n$, then *block iterative methods* can become attractive alternatives to the basic *point* iterative methods. These essentially involve the solution of N systems of equations of size $n_i,\ i = 1, 2, \ldots, N$ within a *global* iterative structure, such as Gauss-Seidel, for instance: thus the *Block Gauss-Seidel* method. Furthermore, these N systems of equations are nonhomogeneous and have nonsingular coefficient matrices and either direct or iterative methods may be used to solve them. It is not required that the same method be used to solve all the diagonal blocks. Instead, it is possible to tailor methods to the particular block structures.

If a direct method is used, then a decomposition of the diagonal block may be formed once and for all before initializing the global iteration process. In each subsequent global iteration, solving for that block then reduces to a forward and backward substitution operation. The nonzero structure of the blocks may be such that this is a particularly attractive approach. For example, if the diagonal blocks are themselves diagonal matrices, or if they are upper or lower triangular matrices or even tridiagonal matrices, then it is very easy to obtain their *LU* decomposition, and a block iterative method becomes very attractive.

If the diagonal blocks do not possess such a structure, and when they are of large dimension, it may be appropriate to use an iterative method to solve each of the block systems. In this case, we have many inner iterative methods (one per block) within an outer (or global) iteration. A number of tricks may be used to speed up this process. First, the solution computed for any block at global iteration k should be used as the initial approximation to the solution of this same block at iteration $k + 1$. Second, it is hardly worthwhile computing a highly accurate solution in early (outer) iterations. We should require only a small number of digits of accuracy until the global process begins to converge. One convenient way to achieve this is to carry out only a fixed, small number of iterations for each inner solution.

The *IAD — Iterative Aggregation/Disaggregation* methods are related to block iterative methods. They are particularly powerful when the Markov chain is *NCD — Nearly*

Completely Decomposable, as the partitions are chosen based on how strongly the states of the Markov chain interact with one another, [18, 45]. The choice of good partitions for both block iterative methods and IAD methods is an active area of current research.

3.2.4 Projection Methods

Projection methods have begun to be applied successfully to Markov chain problems, [51]. Whereas iterative methods begin with an approximate solution vector that is modified at each iteration and which (supposedly) converges to a solution, projection methods create vector subspaces and search for the best possible approximation to the solution that can be obtained from that subspace. With a given subspace, for example, it is possible to extract a vector $\hat{\pi}$ that is a linear combination of a set of basis vector for that space and which minimizes $\|\hat{\pi}Q\|$ in some vector norm. This vector $\hat{\pi}$ may then be taken as an approximation to the solution of $\pi Q = 0$. This is the basis for the *GMRES, Generalized Minimal Residual* algorithm. Another popular projection method is the method of *Arnoldi*. The subspace most often used is the Krylov subspace, $K_m = span\{v_1, v_1Q, \ldots, v_1Q^{m-1}\}$, constructed from a starting vector v_1 and successive iterates of the power method. The computed vectors are then orthogonalized with respect to one another. It is also possible to construct "iterative" variants of these methods. When the subspace reaches some maximum size, the best approximation is chosen from this subspace and a new subspace generated using this approximation as the initial starting point.

Preconditioning techniques are frequently used to improve the convergence rate of iterative Arnoldi and GMRES. This typically amounts to replacing the original system $\pi Q = 0$ by

$$\pi Q M^{-1} = 0,$$

where M is a matrix whose inverse is easy to compute. The objective of preconditioning is to modify the system of equations to obtain a coefficient matrix with a fast rate of convergence. It is worthwhile pointing out that preconditioning may also be used with the basic power method to improve its rate of convergence. The inverse of the matrix M is generally computed from an *Incomplete LU factorization* of the matrix Q. These preconditioning techniques can also be applied to iterative methods.

3.3 The Computation of Transient Distributions

There exists several numerical techniques for obtaining transient solutions of homogeneous, irreducible Markov chains. These techniques are based either on computing matrix exponentials or integrating the Chapman-Kolmogorov system of differential equations:

$$\begin{cases} \dfrac{d\pi(t)}{dt} = \pi(t)Q, \; t \in [0,\; T] \\ \pi(0) \quad = \pi_0 \qquad \text{an initial probability distribution.} \end{cases} \tag{5}$$

The transient distribution, $\pi(t)$, is the solution of (5) and is known to be given by

$$\pi(t) = e^{Qt}\pi_0.$$

Since the matrix exponential is full even when the original matrix is sparse, the practical computation of e^{Qt} in full remains possible only when Q is relatively small, i.e., when the number of states in the Markov chain does not exceed a few hundred. In [47], Moler and Van Loan provide an instructive review of possible methods applicable in this context. Although this review shows that none of the methods are unconditionally acceptable for all classes of problems, methods such as those of the Padé-type or matrix decompositions, with careful implementation, can be satisfactory in many contexts. These methods involve matrix-matrix operations.

To address large problems, the family of series methods, in which matrix-vector operations are paramount, appears to be a reasonable choice. One method of this class, the *uniformization* method is particularly widely used. The uniformization (or randomization) technique is based on the evaluation of the pth partial Taylor series expansion of the matrix exponential, [29, 31]. The length p is fixed so that a prescribed tolerance on the approximation is satisfied. Since Q is essentially nonnegative (i.e., the diagonal elements of Q are negative and the off-diagonal elements are nonnegative), a naive use of the expression $\pi(t) = e^{Qt}\pi_0 \approx \sum_{k=0}^{p}(1/k!)(Qt)^k\pi_0$ is subject to severe roundoff errors due to terms of alternating signs. Uniformization uses the modified formulation $\pi(t) = e^{\alpha(P-I)t}\pi_0 = e^{-\alpha t}e^{\alpha Pt}\pi_0$ where $\alpha \equiv \max_i |q_{ii}|$ and $P \equiv \frac{1}{\alpha}Q + I$ is nonnegative with $\|P\|_1 = 1$. The resulting truncated approximation

$$\tilde{\pi}(t) = \sum_{k=0}^{p} e^{-\alpha t}\frac{(\alpha t)^k}{k!}\pi_0 P^k$$

involves only nonnegative terms and becomes numerically stable. The popularity of Unifomization is due to three reasons. Firstly, its handiness and malleability facilitate its implementation – only a matrix-vector product is needed per iteration. Secondly, the transformation from Q to P has a probabilistic interpretation. Thirdly and perhaps most important, it works surprizingly well in a great variety of circumstances.

A different class of method, that of ordinary differential equation (ODE) solvers, is appealing because of the high availability of ready-to-use efficient library routines for solving initial value ODE problems. In addition to being multiple- or single-step, ODE solvers can be explicit or implicit, yielding four possible categories. Each category yields new classes of methods in their own right — depending on their derivation, their analytic and numerical properties, or on implementation aspects. For example, some of the particularities of a method can be: multistage or otherwise, order, stability region, matrix-free or otherwise, stiff or non-stiff, computational cost, etc. In general, implicit methods — which are more costly, appear suitable for stiff-problems while cheap explicit methods are satisfactory only on non-stiff problems. Recently, Krylov projection-type methods have been developed and these appear to be particularly useful for Markov chain problems. The Krylov-based algorithm generates an approximation to $\pi(t) = \exp(Qt)\pi_0$ and computes the matrix exponential times a vector rather than the matrix exponential in isolation. The underlying principle is to approximate

$$\pi(t) = e^{Qt}\pi_0 = \pi_0 + \frac{(Qt)}{1!}\pi_0 + \frac{(Qt)^2}{2!}\pi_0 + \cdots \tag{6}$$

by an element of the Krylov subspace

$$\mathcal{K}_m(Qt, \pi_0) = \text{Span}\{\pi_0, (Qt)\pi_0, \ldots, (Qt)^{m-1}\pi_0\}, \tag{7}$$

where m, the dimension of the Krylov subspace, is small compared to n, the order of the coefficient matrix (usually $m \leq 50$ whilst n can exceed many hundreds of thousands). The approximation used is

$$\tilde{\pi}(t) = \beta V_{m+1} \exp(\bar{H}_{m+1}\, t)e_1 \tag{8}$$

where $\beta = \|\pi_0\|_2$; $V_{m+1} = [\pi_{01}, \ldots, \pi_{0m+1}]$ and $\bar{H}_{m+1} = [h_{ij}]$ are, respectively, the orthonormal basis and the upper Hessenberg matrix resulting from the well-known Arnoldi process (see, e.g., [28, 63]); e_1 is the first unit basis vector. The distinctive feature is that the original large problem (6) is converted to the small problem (8) which is more desirable. Within the framework of Markov chains, relevant studies are those of Philippe and Sidje, [52], and Sidje, [68, 69].

Systematic and extensive numerical comparisons of various methods for computing matrix exponentials in general, and transient solutions of Markov chains in particular are modest in the literature. One can mention the attempt made in Sidje, [68], where ODE solvers from the NAG library were used. However, the assessment provided is debatable because, with regard to other ODE libraries, the efficiency of the ODE chapter of NAG is questioned. In Clarotti, [19], a brief description of a customized implicit-type method is outlined. Unfortunately no experiments nor comparisons, are reported that confirm or deny the superiority of the approach. The work in Reibman and Trivedi, [61], later continued in Malhorta and Trivedi, [42], are also worth mentioning, for the ODE solution techniques used therein have been tailored specifically for the Markovian context. In [42] for instance, a comprehensive analysis of the issues (namely, largeness, stiffness and accuracy) faced when solving Markov chains numerically is presented and a comparison of four different solution techniques is undertaken. The comparison suggests that uniformization is best on non-stiff problems but is inferior to implicit ODE-solvers, such as the implicit third-order RK method, on stiff-problems.

3.4 Markov Chains with Kronecker Product Form

In the recent past, much attention has been given to Markov chains in which the transition matrix can be written as a sum of Kronecker products [10, 21, 25, 34, 53, 54, 55]. Such matrix representations provide a means of performing Markov chain modelling without the problem of having to store huge transition matrices. A prime example is a *Stochastic Automata Network* (SAN). A SAN consists of a number of individual stochastic automata that operate more or less independently of each other. Each individual automaton is represented by a number of states and rules that govern the manner in which it moves from one state to the next. The state of an automaton at any time t is just the state it occupies at time t and the state of the SAN at time t is given by the state of each of its constituent automata. An automaton may be thought of as a component in a Markov chain state descriptor. It has been observed that SANs provide a natural means of describing parallel and distributed systems since such systems are

often viewed as collections of components that operate more or less independently, requiring only infrequent interaction such as synchronizing their actions, or operating at different rates depending on the state of parts of the overall system. This is exactly the viewpoint adopted by SANs.

The compact form in which the transition matrix that characterizes the model is kept helps keep memory requirements within manageable limits and avoids the state space explosion associated with other state based approaches. Therefore, the state space explosion problem associated with Markov chain models is mitigated by the fact that the state transition matrix is not stored, nor even generated. Instead, it is represented by a number of much smaller matrices and from these all relevant information may be determined without explicitly forming the global matrix. The implication is that a considerable saving in memory is effected by keeping the infinitesimal generator in this fashion. A potential source of memory waste with the Kronecker-based approach is due to the fact that the tensor product state space can become much larger than the actual model state space. While this is not a problem for storing the transition matrix itself (since it is not stored) it can pose a problem for storing the vectors needed to compute numerical solutions. Research by Ciardo, [17], and others have produced techniques that go a long way towards eliminating this problem.

In order to benefit from this compact form, the descriptor is never expanded into a single large matrix. Consequently, all subsequent operations must necessarily work with the model in its descriptor form and hence numerical operations on the underlying Markov chain infinitesimal generator become more costly. Previously, this cost was sufficiently high to discourage the application of Kronecker-based technologies. Recent results will most likely change this situation. These show how the application of successive modelling strategems and numerical savoir-faire reduce the time needed to compute stationary distributions by several orders of magnitude, thereby reducing considerably this perceived disadvantage. In particular, the essential role that funtional transitions play in this scenario needs to be emphasized. Functional transitions allow a system modeled as a SAN to use fewer automata and fewer synchronizing transitions. In other words, if functional transitions cannot be handled by the modelling techniques used, then a given system can be modeled as a SAN only if additional automata are included and these automata linked to others by means of synchronizing transitions.

Formally, a SAN is a set of automata whose dynamic behavior is governed by a set of *events*. Events are said to be *local* if they provoke a transition in a single automaton, and *synchronizing* if they provoke a transition in more than one automaton. It goes without saying that a single event can generate more than one transition. A transition that results from a synchronizing event is said to be a *synchronized transition*; otherwise it is called a *local transition*. We denote the number of states in automaton i by n_i and denote by N the number of automata in the SAN.

The behavior of each automaton, $\mathcal{A}^{(i)}$, for $i = 1, \ldots, N$, is described by a set of square matrices, all of order n_i. In our context, a SAN is studied as a continuous-time Markov chain. The rate at which event transitions occur may be constant or may depend upon the state in which they take place. In this last case they are said to be functional (or state-dependent). Synchronized transitions may be functional or non-functional. Functional transitions allow a system to be modeled as a SAN using fewer automata and

fewer synchronizing transitions. In other words, if functional transitions cannot be handled by the modelling techniques used, then a given system may be modeled as a SAN if additional automata are included and these automata are linked to others by means of synchronizing transitions.

In the absence of synchronizing events and functional transitions, the matrices which describe $\mathcal{A}^{(i)}$ reduce to a single infinitesimal generator matrix, $Q^{(i)}$, and the global Markov chain generator may be written as

$$Q = \bigoplus_{i=1}^{N} Q^{(i)} = \sum_{i=1}^{N} I_{n_1} \otimes \cdots \otimes I_{n_{i-1}} \otimes Q^{(i)} \otimes I_{n_{i+1}} \otimes \cdots \otimes I_{n_N}. \tag{9}$$

The tensor sum formulation is a direct result of the independence of the automata, and the formulation as a sum of tensor products, a result of the defining property of tensor sums. The probability distribution at any time t of this independent N-dimensional system is known to be

$$\pi(t) = \bigotimes_{i=1}^{N} \pi^{(i)}(t). \tag{10}$$

Now consider the case of SANs which contain synchronizing events but no functional transitions and let us denote by $Q_l^{(i)}$, $i = 1, 2, \ldots, N$, the matrix consisting only of the transitions that are local to $\mathcal{A}^{(i)}$. Then, the part of the global infinitesimal generator that consists uniquely of local transitions may be obtained by forming the tensor sum of the matrices $Q_l^{(1)}, Q_l^{(2)}, \ldots, Q_l^{(N)}$. As is shown in [53], stochastic automata networks may always be treated by separating out the local transitions, handling these in the usual fashion by means of a tensor sum and then incorporating the sum of two additional tensor products per synchronizing event. The first of these two additional tensor products may be thought of as representing the actual synchronizing event and its rates, and the second corresponds to an updating of the diagonal elements in the infinitesimal generator to reflect these transitions. Equation (9) becomes

$$Q = \bigoplus_{i=1}^{N} Q_l^{(i)} + \sum_{e \in \mathcal{E}} \left(\bigotimes_{i=1}^{N} Q_{e^+}^{(i)} + \bigotimes_{i=1}^{N} Q_{e^-}^{(i)} \right). \tag{11}$$

Here $\mathcal{E}$ is the set of synchronizing events. Furthermore, since tensor sums are defined in terms of a matrix sum of tensor products, the infinitesimal generator of a system containing N stochastic automata with E synchronizing events (and no functional transition rates) may be written as

$$Q = \sum_{j=1}^{2E+N} \bigotimes_{i=1}^{N} Q_j^{(i)}. \tag{12}$$

This formula is referred to as the *descriptor* of the stochastic automata network. Efficient algorithms to form the product of this descriptor and a vector, which is the fundamental component in the numerical procedures used to analyze SANs, have been developed and are discussed in [23, 4]

Let us momentarily return to equation (11) to consider how best to handle the diagonal elements of Q. Since the numerical methods used to compute solutions of SANs are usually iterative, the most important operation is that of multiplying the descriptor with a vector and hence it is essential to keep the cost of this multiplication to a minimum. One way to reduce costs is to precompute the *diagonal of the descriptor*. In this case, the descriptor may be considered as being composed of two parts:

- D, a vector containing the diagonal of the descriptor
- $\bar{Q}$, the descriptor itself with the exception that all the diagonal elements of the matrices of each tensor product term are set to zero.

From a practical point of view, this is most easily accomplished by setting all the diagonal elements of local matrices $Q_l^{(i)}$ to zero. In each tensor product term corresponding to a synchronizing event e, the diagonal elements of the matrices corresponding to the automaton which "owns" this event must also be set to zero. A second advantage of this astuce now becomes apparent. Normally each synchronizing event generates two tensor product terms. The first term contains the rates of occurrence of the synchronizing event and the second contains exclusively the elements with which to adjust the diagonal. This second term, once computed, only generates elements on the diagonal of the descriptor. Precomputing the diagonal therefore allows us to eliminate the second tensor product term of each synchronizing event thereby reducing the number of terms requiring manipulation during the multiplication phase. The diagonal elements arising from synchronization terms are added to the diagonal elements corresponding to the tensor sum part of the descriptor and the number of tensor product terms in the descriptor is reduced from $2E + N$ to $E + N$. Precomputing the diagonal brings about even greater savings when the descriptor has functional elements, since the evaluation of functions on the diagonal must all be precomputed, but only once.

Precomputation of the diagonal does have a cost associated with it, although this cost manifests itself only once, namely, during the preparation of the descriptor. On the other hand, the benefits derived from this approach occur each time a vector–descriptor product is computed. A second disadvantage of this approach is the necessity of storing the diagonal elements themselves. Since the representation of the matrix is extremely small, this has the effect of almost doubling the amount of memory needed. However, the augmentation in memory use nevertheless remains low compared to the needs of storing the entire matrix using sparse matrix technology. To counterbalance these inconveniences, this approach provides rapid access to the diagonal of the descriptor with the resulting advantages of

- Easy computation of the largest element of the descriptor, since, given that the descriptor is a representation of the infinitesimal generator, the largest element will always be found along the diagonal.
- Ease of use for implementing certain preconditioning techniques, which, as we shall see later, require access to the diagonal.

Finally, let us consider the effect of introducing functional transitions into SANs. It should be apparent that the introduction of functional transition rates has no effect on the *structure* of the global transition rate matrix other than when functions evaluate to zero in which case a degenerate form of the original structure is obtained. In other

words, the placement of zero versus nonzero elements essentially remains unchanged. What may change is the value of nonzero elements. The nonzero structure of the SAN descriptor is just as before (except in the case when a function evaluates to zero but even here, the sparse data structures used need not be altered). However, because of possible value changes, the usual tensor operations are no longer valid. Since regular tensor products are unable to handle functional transitions it is necessary to use a *Generalized Tensor Algebra*, (GTA) [23], to overcome this difficulty. In particular, this GTA provides some associativity, commutativity, distributivity and compatibility over multiplication properties that enable the descriptor of a SAN with synchronizing events and functional transitions to be handled with algorithms almost identical to those of SANs with no functional transitions. Recent and ongoing research concerns the development of procedures destined to minimize memory requirements, [4], and the computational burden of applying the SAN modelling concepts, [23], as well as extending the applicability of the modelling procedure to incorporate phase-type distributions, [64].

3.5 Structured Markov Chains of M/G/1- and GI/M/1-Type

Much work has been carried out by Neuts and his colleagues, [49, 50, 59, 60], on the numerical solution of Markov chains whose transition matrices have a special block structure — a block structure that arises frequently when modeling queueing systems. In the simplest case, these matrices are infinite block tridiagonal matrices in which the three diagonal blocks repeat after some initial period. To capture this non-zero structure, we write such a matrix as

$$\begin{pmatrix} B_{00} & B_{01} & 0 & 0 & 0 & 0 & \cdots \\ B_{10} & A_1 & A_2 & 0 & 0 & 0 & \cdots \\ 0 & A_0 & A_1 & A_2 & 0 & 0 & \cdots \\ 0 & 0 & A_0 & A_1 & A_2 & 0 & \cdots \\ & & & \ddots & \ddots & \ddots & \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{pmatrix} \tag{13}$$

in which all submatrices, $A_j,\ j\ =\ 0,\ 1,\ 2,\ ...$ are square and have the same dimension; the matrix B_{00} is also square and need not have the same size as A_1 and the dimensions of B_{01} and B_{10} are defined to be in accordance with the dimensions of B_{00} and A_1. A transition matrix having this structure arises when each state of the Markov chain can be written as a pair $\{(\eta, k), \eta \geq 0, 1 \leq k \leq K\}$ and the states ordered, first according to increasing value of the parameter η and for states with the same η value, by increasing value of k.This has the effect of grouping the states into "levels" according to their η value. The block tridiagonal effect is achieved when transitions are permitted only between states of the same level (diagonal blocks),to states in the next highest level (super-diagonal blocks), and to states in the adjacent lower level (sub-diagonal blocks). The repetitive nature of the blocks themselves arises if, after boundary conditions are taken into consideration (which gives the initial blocks $B_{00},\ B_{01}$ and B_{10}) the transition rates/probabilities are constant from level to level. A Markov chain whose transition matrix has this block tridiagonal structure is said to belong to the class of *Quasi-Birth-Death* (QBD) processes.

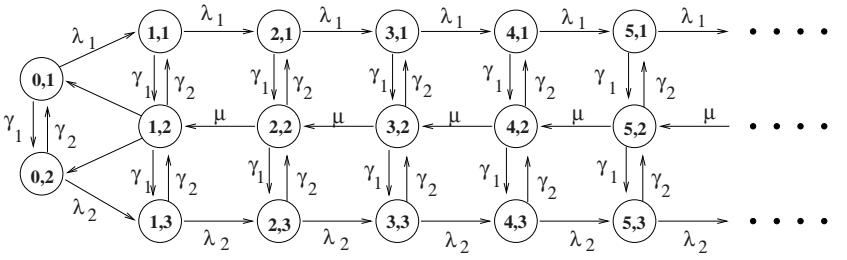


Fig. 1. State transition diagram for an M/M/1-type process

3.5.1 Quasi-birth-Death Processes

Consider the Markov chain whose state transition diagram is shown in Figure 1.

Its transition rate matrix is

$$Q = \begin{pmatrix}
* & \gamma_1 & \lambda_1 & & & & & & & \\
\gamma_2 & * & & \lambda_2 & & & & & & \\
\hline
\multicolumn{2}{c|}{\multirow{2}{*}{$\mu/2\ \mu/2$}} & * & \gamma_1 & \lambda_1 & & & & & \\
\multicolumn{2}{c|}{} & \gamma_2 & * & \gamma_1 & & & & & \\
\multicolumn{2}{c|}{} & & \gamma_2 & * & \lambda_2 & & & & \\
\hline
\multicolumn{2}{c|}{} & \multicolumn{2}{c|}{\multirow{2}{*}{μ}} & * & \gamma_1 & \lambda_1 & & & \\
\multicolumn{2}{c|}{} & \multicolumn{2}{c|}{} & \gamma_2 & * & \gamma_1 & & & \\
\multicolumn{2}{c|}{} & \multicolumn{2}{c|}{} & & \gamma_2 & * & \lambda_2 & & \\
\hline
\multicolumn{2}{c|}{} & \multicolumn{2}{c|}{} & \multicolumn{2}{c|}{\multirow{2}{*}{μ}} & * & \gamma_1 & \lambda_1 & \\
\multicolumn{2}{c|}{} & \multicolumn{2}{c|}{} & \multicolumn{2}{c|}{} & \gamma_2 & * & \gamma_1 & \\
\multicolumn{2}{c|}{} & \multicolumn{2}{c|}{} & \multicolumn{2}{c|}{} & & \gamma_2 & * & \lambda_2 \\
\hline
\multicolumn{2}{c|}{} & \multicolumn{2}{c|}{} & \multicolumn{2}{c|}{} & \multicolumn{2}{c|}{\multirow{2}{*}{μ}} & * & \gamma_1 \\
\multicolumn{2}{c|}{} & \multicolumn{2}{c|}{} & \multicolumn{2}{c|}{} & \multicolumn{2}{c|}{} & \gamma_2 & * \\
\hline
\multicolumn{2}{c|}{} & \multicolumn{2}{c|}{} & \multicolumn{2}{c|}{} & \multicolumn{2}{c|}{} & \ddots & \ddots
\end{pmatrix}$$

and has the typical block tridiagonal structure which makes it an ideal candidate for solution by the matrix geometric method. Its diagonal elements, marked by asterisks, are such that the sum across each row is zero. We have the following block matrices

$$A_0 = \begin{pmatrix} 0 & 0 & 0 \\ 0 & \mu & 0 \\ 0 & 0 & 0 \end{pmatrix}, A_1 = \begin{pmatrix} -(\gamma_1+\lambda_1) & \gamma_1 & 0 \\ \gamma_2 & -(\mu+\gamma_1+\gamma_2) & \gamma_1 \\ 0 & \gamma_2 & -(\gamma_2+\lambda_2) \end{pmatrix}, A_2 = \begin{pmatrix} \lambda_1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & \lambda_2 \end{pmatrix}$$

and

$$B_{00} = \begin{pmatrix} -(\gamma_1+\lambda_1) & \gamma_1 \\ \gamma_2 & -(\gamma_2+\lambda_2) \end{pmatrix}, \quad B_{01} = \begin{pmatrix} \lambda_1 & 0 & 0 \\ 0 & 0 & \lambda_2 \end{pmatrix}, \quad B_{10} = \begin{pmatrix} 0 & 0 \\ \mu/2 & \mu/2 \\ 0 & 0 \end{pmatrix}.$$

The matrix geometric method derives relationships which involve these blocks and which permits the stationary distribution to be efficient computed. Since this procedure also applies to more general GI/M/1-type processes, we defer considering it until the next section. The most common extensions to this simplest case are matrices which are block upper Hessenberg (referred to as $M/G/1$ type and solved using the matrix analytic approach) and those that are block lower Hessenberg (referred to as $GI/M/1$ type, solved using the matrix geometric approach). Results are also available for the case in which parameters η and k are both finite, which results in a finite Markov chain as well as for the case in which transitions may be dependent on the level η.

3.5.2 Structured Markov Chains of the GI/M/1-Type

In a lower Hessenberg matrix A, all elements a_{ij} must be zero for values of $j > i + 1$. In other words, if moving from top to bottom, we designate the three diagonals of a tridiagonal matrix as the super-diagonal, the diagonal and the subdiagonal, then a lower Hessenberg matrix can have non-zero elements only on and below the superdiagonal. For example, the following matrix is a 6×6 lower Hessenberg matrix.

$$A = \begin{pmatrix} a_{00} & a_{01} & 0 & 0 & 0 & 0 \\ a_{10} & a_{11} & a_{12} & 0 & 0 & 0 \\ a_{20} & a_{21} & a_{22} & a_{23} & 0 & 0 \\ a_{30} & a_{31} & a_{32} & a_{33} & a_{34} & 0 \\ a_{40} & a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \\ a_{50} & a_{51} & a_{52} & a_{53} & a_{54} & a_{55} \end{pmatrix}$$

Block lower Hessenberg matrices, found in GI/M/1-type stochastic processes, are just the block counterparts of lower Hessenberg matrices. In a similar manner we can define block *upper* Hessenberg matrices, which are found in M/G/1-type stochastic processes, as block matrices whose only nonzero blocks are on or above the diagonal blocks and along the subdiagonal block. One is essentially the block transpose of the other. This leads us to caution the reader about the notation used to designate the individual blocks. With block lower Hessenberg matrices, the non-zero blocks are numbered from right to left as A_0, A_1, $A_2, \ldots$, while in block upper Hessenberg matrices they are numbered in the reverse order, from left to right. As we have just seen, we choose to denote the blocks in QBD processes (which are simultaneously both upper and lower block Hessenberg) from left to right, i.e., using the block upper Hesssenberg notation. Our interest in this section is with Markov chains whose infinitesimal generators Q have the following repetitive block lower Hessenberg structure.

$$Q = \begin{pmatrix} B_{00} & B_{01} & 0 & 0 & 0 & 0 & 0 & \cdots \\ B_{10} & B_{11} & A_0 & 0 & 0 & 0 & 0 & \cdots \\ B_{20} & B_{21} & A_1 & A_0 & 0 & 0 & 0 & \cdots \\ B_{30} & B_{31} & A_2 & A_1 & A_0 & 0 & 0 & \cdots \\ B_{40} & B_{41} & A_3 & A_2 & A_1 & A_0 & 0 & \cdots \\ \vdots & & \ddots & \ddots & \ddots & \ddots & \ddots & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \cdots \end{pmatrix}$$

Markov chains with this structure occur when the states are grouped into levels, similar to that for QBD processes, but now transitions are no longer confined to inter-level and to adjacent neighboring levels; transitions are also permitted from any level to any *lower* level.

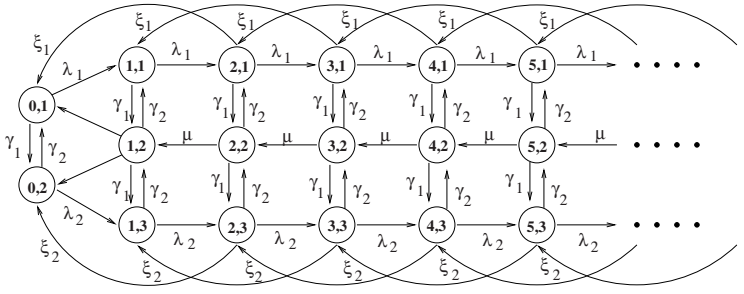


Fig. 2. State transition diagram for a GI/M/1-type process

For example, the Markov chain of Example 3.5.1, modified so that it incorporates additional transitions ξ_1 and ξ_2 to lower non-neighboring states is shown in Figure 2. Its transition matrix is given by

$$Q = \left(\begin{array}{c|cc|ccc|ccc|cc|c}
* & \gamma_1 & \lambda_1 & & & & & & & & & \\
\gamma_2 & * & & \lambda_2 & & & & & & & & \\
\hline
\multicolumn{2}{c|}{\multirow{2}{*}{$\mu/2\ \mu/2$}} & * & \gamma_1 & \lambda_1 & & & & & & & \\
\multicolumn{2}{c|}{} & \gamma_2 & * & \gamma_1 & & & & & & & \\
\multicolumn{2}{c|}{} & & \gamma_2 & * & \lambda_2 & & & & & & \\
\hline
\xi_1 & & & * & \gamma_1 & & \lambda_1 & & & & & \\
 & & \mu & \gamma_2 & * & \gamma_1 & & & & & & \\
 & \xi_2 & & & \gamma_2 & * & & \lambda_2 & & & & \\
\hline
 & \xi_1 & & & & & * & \gamma_1 & & \lambda_1 & & \\
 & & & & \mu & & \gamma_2 & * & \gamma_1 & & & \\
 & & \xi_2 & & & & & \gamma_2 & * & & \lambda_2 & \\
\hline
 & & & \xi_1 & & & & & & * & \gamma_1 & \lambda_1 \\
 & & & & & & \mu & & & \gamma_2 & * & \gamma_1 \\
 & & & & & \xi_2 & & & & & \gamma_2 & * \\
\hline
 & & & & & & \ddots & & & \ddots & \ddots & \ddots
\end{array}\right)$$

In this case, we show two boundary columns (B_{i0} and B_{i1}, i = 0, 1, 2, ...). In some applications, such as queueing systems with bulk arrivals, more than two boundary columns can occur and this may necessitate a restructuring of the matrix. Consider, for example the generator matrix

$$Q = \begin{pmatrix}
B_{00} & B_{01} & B_{02} & A_0 & & & & & & & & \\
B_{10} & B_{11} & B_{12} & A_1 & A_0 & & & & & & & \\
B_{20} & B_{21} & B_{22} & A_2 & A_1 & A_0 & & & & & & \\
B_{30} & B_{31} & B_{32} & A_3 & A_2 & A_1 & A_0 & & & & & \\
B_{40} & B_{41} & B_{42} & A_4 & A_3 & A_2 & A_1 & A_0 & & & & \\
B_{50} & B_{51} & B_{52} & A_5 & A_4 & A_3 & A_2 & A_1 & A_0 & & & \\
B_{60} & B_{61} & B_{62} & A_6 & A_5 & A_4 & A_3 & A_2 & A_1 & A_0 & & \\
B_{70} & B_{71} & B_{72} & A_7 & A_6 & A_5 & A_4 & A_3 & A_2 & A_1 & A_0 & \\
B_{80} & B_{81} & B_{82} & A_8 & A_7 & A_6 & A_5 & A_4 & 0 & 0 & A_1 & A_0 \\
\vdots & \vdots & \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots
\end{pmatrix}$$

At present, this matrix is *not* block lower Hessenberg. However, if it is restructured into the form

$$\begin{pmatrix}
\overline{B_{00}} & \overline{A_0} & 0 & 0 & \cdots \\
\overline{B_{10}} & \overline{A_1} & \overline{A_0} & 0 & \cdots \\
\overline{B_{20}} & \overline{A_2} & \overline{A_1} & \overline{A_0} & \\
\vdots & & \ddots & \ddots & \ddots
\end{pmatrix} = \left(\begin{array}{ccc|ccc|ccc|c}
B_{00} & B_{01} & B_{02} & A_0 & & & & & & \\
B_{10} & B_{11} & B_{12} & A_1 & A_0 & & & & & \\
B_{20} & B_{21} & B_{22} & A_2 & A_1 & A_0 & & & & \\
\hline
B_{30} & B_{31} & B_{32} & A_3 & A_2 & A_1 & A_0 & & & \\
B_{40} & B_{41} & B_{42} & A_4 & A_3 & A_2 & A_1 & A_0 & & \\
B_{50} & B_{51} & B_{52} & A_5 & A_4 & A_3 & A_2 & A_1 & A_0 & \\
\hline
B_{60} & B_{61} & B_{62} & A_6 & A_5 & A_4 & A_3 & A_2 & A_1 & A_0 \\
B_{70} & B_{71} & B_{72} & A_7 & A_6 & A_5 & A_4 & A_3 & A_2 & A_1 \; A_0 \\
B_{80} & B_{81} & B_{82} & A_8 & A_7 & A_6 & A_5 & A_4 & 0 & 0 \; A_1 \; A_0 \\
\hline
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & & \ddots & \ddots & \ddots \; \ddots \; \ddots & \ddots
\end{array}\right)$$

with

$$\overline{A_0} = \begin{pmatrix} A_0 & & \\ A_1 & A_0 & \\ A_2 & A_1 & A_0 \end{pmatrix}, \quad \overline{A_1} = \begin{pmatrix} A_3 & A_2 & A_1 \\ A_4 & A_3 & A_2 \\ A_5 & A_4 & A_3 \end{pmatrix}, \quad \overline{B_{00}} = \begin{pmatrix} B_{00} & B_{01} & B_{02} \\ B_{10} & B_{11} & B_{12} \\ B_{20} & B_{21} & B_{22} \end{pmatrix}, \quad \cdots$$

then the matrix $\overline{Q}$ does have the desired block lower Hessenberg form and the matrix geometric techniques which we now outline may be successfully applied to it.

Our objective is to compute the stationary probability vector π from the system of equations $\pi Q = 0$. Let π be partitioned conformally with Q, i.e. $\pi = (\pi_0, \pi_1, \pi_2, \cdots)$ where $\pi_i = (\pi(i,1), \pi(i,2), \cdots \pi(i,K))$ for $i = 0, 1, \cdots$ and $\pi(i,k)$ is the probability of finding the system in state (i,k) under steady state conditions. Neuts has shown that there exists a matrix geometric solution to this problem, that there exists a positive matrix R such that

$$\pi_i = \pi_{i-1} R, \quad \text{for } i = 2, 3, \ldots$$

i.e., that

$$\pi_i = \pi_1 R^{i-1}, \quad \text{for } i = 2, 3, \ldots$$

Observe that from $\pi Q = 0$ we have

$$\sum_{k=0}^{\infty} \pi_{k+j} A_k = 0, \quad j = 1, 2, \ldots$$

and in particular,

$$\pi_1 A_0 + \pi_2 A_1 + \sum_{k=2}^{\infty} \pi_{k+1} A_k = 0$$

Substituting $\pi_i = \pi_1 R^{i-1}$

$$\pi_1 A_0 + \pi_1 R A_1 + \sum_{k=2}^{\infty} \pi_1 R^k A_k = 0$$

or

$$\pi_1 \left(A_0 + R A_1 + \sum_{k=2}^{\infty} R^k A_k \right) = 0$$

which provides us the following mechanism by which the matrix R may be computed.

$$A_0 + R A_1 + \sum_{k=2}^{\infty} R^k A_k = 0 \tag{14}$$

When $A_k = 0$ for $k > 2$, we obtain the QBD case. Rearranging Equation (14), we find

$$R = -A_0 A_1^{-1} - \sum_{k=2}^{\infty} R^k A_k A_1^{-1}$$

which leads to the iterative procedure

$$R_{(0)} = 0; \quad R_{(l+1)} = -A_0 A_1^{-1} - \sum_{k=2}^{\infty} R_{(l)}^k A_k A_1^{-1}, \ \ l = 1, 2, \ldots$$

which Neuts has shown to be non decreasing and converges to the matrix R. In many cases, the structure of the infinitesimal generator is such that the blocks A_i are zero for relatively small values of i, which limits the computational effort needed in each iteration. The number of iterations needed for convergence using this approach is frequently large. Fortunately more efficient but also more complex algorithms, have been developed and may be found in the current literature.

We now turn to the derivation of the initial subvectors π_0 and π_1. From the first equation of $\pi Q = 0$, we have

$$\sum_{i=0}^{\infty} \pi_i B_{i0} = 0$$

and we may write

$$\pi_0 B_{00} + \sum_{i=1}^{\infty} \pi_i B_{i0} = \pi_0 B_{00} + \sum_{i=1}^{\infty} \pi_1 R^{i-1} B_{i0} = \pi_0 B_{00} + \pi_1 \left(\sum_{i=1}^{\infty} R^{i-1} B_{i0} \right) = 0, \tag{15}$$

while from the second equation of $\pi Q = 0$,

$$\pi_0 B_{01} + \sum_{i=1}^{\infty} \pi_i B_{i1} = 0, \quad \text{i.e.,} \quad \pi_0 B_{01} + \pi_1 \sum_{i=1}^{\infty} R^{i-1} B_{i1} = 0. \tag{16}$$

Putting Equations (15) and (15) together in matrix form, we see that we can compute π_0 and π_1 from

$$(\pi_0, \pi_1) \begin{pmatrix} B_{00} & B_{01} \\ \sum_{i=1}^{\infty} R^{i-1} B_{i0} & \sum_{i=1}^{\infty} R^{i-1} B_{i1} \end{pmatrix} = (0, 0).$$

The computed values of π_0 and π_1 must now be normalized by dividing them by

$$\alpha = \pi_0 e + \pi_1 \left(\sum_{i=1}^{\infty} R^{k-1} \right) e = \pi_0 e + \pi_1 (I - R)^{-1} e.$$

In the case of discrete-time Markov chains, as opposed to the continuous-time case just outlined, it suffices to replace $-A_1^{-1}$ with $(I - A_1)^{-1}$, as we described for QBD processes.

3.5.3 Structured Markov Chains of the M/G/1-Type

We now move to block upper-Hessenberg Markov chains, also called M/G/1-type processes. For example, the state transition diagram of the Markov chain of Example 3.5.1 now modified so that it incorporates additional transitions ζ_1 and ζ_2 to higher numbered non-neighboring states is shown in Figure 3. The matrix geometric method is not applicable to M/G/1-type Markov chains. Instead, Neuts has developed a matrix analytic method which we now discuss

In the past two sections concerning QBD and GI/M/1-type processes, we posed the problem in terms of continuous-time Markov chains. Discrete-time Markov chains can be treated if the matrix inverse A_1^{-1} is replaced with the inverse $(I - A_1)^{-1}$ where it is understood that A_1 is taken from a stochastic matrix. This time we shall consider the discrete-time case. Specifically, we consider the case when the stochastic transition probability matrix is irreducible and has the structure

$$P = \begin{pmatrix} B_{00} & B_{01} & B_{02} & B_{03} & \cdots & B_{0j} & \cdots \\ B_{10} & A_1 & A_2 & A_3 & \cdots & A_j & \cdots \\ 0 & A_0 & A_1 & A_2 & \cdots & A_{j-1} & \cdots \\ 0 & 0 & A_0 & A_1 & \cdots & A_{j-2} & \cdots \\ 0 & 0 & 0 & A_0 & \cdots & A_{j-3} & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{pmatrix}$$

in which all submatrices, A_j, $j = 0, 1, 2, \ldots$ are square and of order K, and B_{00} is square but not necessarily of order K. Be aware that whereas previously the block A_0 was the right-most non-zero block of our global matrix, this time it is the left-most non-zero block. Notice that the matrix $A = \sum_{i=0}^{\infty} A_i$ is a stochastic matrix. We shall further assume that A is irreducible, the commonly observed case in practice. Instances in which A is not irreducible are treated by Neuts [50]. Since A is finite and irreducible, it has a stationary distribution that we denote by π_A, i.e.,

$$\pi_A A = \pi_A, \quad \text{and} \quad \pi_A e = 1.$$

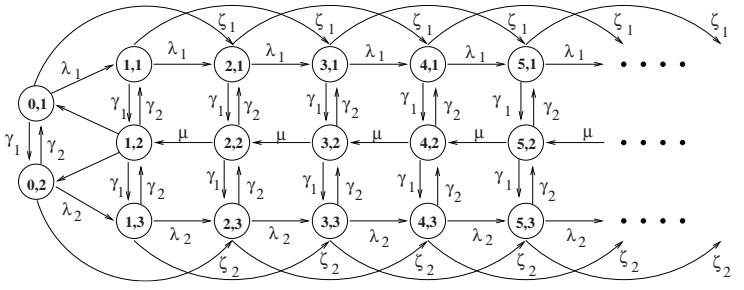


Fig. 3. State transition diagram for an M/G/1-type process

The Markov chain P is known to be positive recurrent if the following condition holds

$$\pi_A \left(\sum_{i=1}^{\infty} i A_i \, e \right) \equiv \pi_A \, b \; < 1. \tag{17}$$

Our objective is the computation of the stationary probability vector π from the system of equations $\pi P = \pi$. As before, we partition π conformally with P, i.e.

$$\pi = (\pi_0, \pi_1, \pi_2, \cdots)$$

where

$$\pi_i = (\pi(i,1), \pi(i,2), \cdots \pi(i,K))$$

for $i = 0, 1, \cdots$ and $\pi(i,k)$ is the probability of finding the system in state (i,k) under steady state conditions. The analysis of M/G/1-type processes is more complicated than that of QBD or GI/M/1-type processes because the subvectors π_i no longer have a matrix geometric relationship with one another.

The key to solving upper block-Hessenberg structured Markov chains is the computation of a certain matrix G which is stochastic if the Markov chain is recurrent, which we assume to be the case. This matrix G has an important probabilistic interpretation. ITs element G_{ij} is the conditional probability that starting in state i of any level $n \geq 2$, the process enters level $n-1$ for the first time by arriving at state j of that level. This matrix satisfies the fixed point equation

$$G = \sum_{i=0}^{\infty} A_i G^i$$

and is indeed is the minimal non-negative solution of

$$X = \sum_{i=0}^{\infty} A_i X^i.$$

It can be found by means of the iteration

$$G_{(0)} = 0; \quad G_{(k+1)} = \sum_{i=0}^{\infty} A_i G_{(k)}^i = 0, \quad k = 0, 1, \ldots$$

Once the matrix G has been computed, then successive components of π can be obtained from a relationship, called Ramaswami's formula, which we now develop. We follow the algebraic approach of Bini and Meini, [8, 46], rather than the original probabilistic approach of Ramaswami, [57]. We begin by writing the system of equations $\pi P = \pi$ as $\pi(I - P) = 0$, i.e.,

$$(\pi_0, \pi_1, \cdots, \pi_j, \cdots) \left(\begin{array}{c|cccccc} I - B_{00} & -B_{01} & -B_{02} & -B_{03} & \cdots & -B_{0j} & \cdots \\ \hline -B_{10} & I - A_1 & -A_2 & -A_3 & \cdots & -A_j & \cdots \\ 0 & -A_0 & I - A_1 & -A_2 & \cdots & -A_{j-1} & \cdots \\ 0 & 0 & -A_0 & I - A_1 & \cdots & -A_{j-2} & \cdots \\ 0 & 0 & 0 & -A_0 & \cdots & -A_{j-3} & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{array}\right) = (0, 0, \cdots 0, \cdots). \tag{18}$$

The submatrix in the lower right block is block Toeplitz. Bini and Meini have shown that there exists a decomposition of this Toeplitz matrix into a block upper triangular matrix U and block lower triangular matrix L with

$$U = \begin{pmatrix} A_1^* & A_2^* & A_3^* & A_4^* & \cdots \\ 0 & A_1^* & A_2^* & A_3^* & \cdots \\ 0 & 0 & A_1^* & A_2^* & \cdots \\ 0 & 0 & 0 & A_1^* & \cdots \\ \vdots & \vdots & \vdots & & \ddots \end{pmatrix} \quad \text{and} \quad L = \begin{pmatrix} I & 0 & 0 & 0 & \cdots \\ -G & I & 0 & 0 & \cdots \\ 0 & -G & I & 0 & \cdots \\ 0 & 0 & -G & I & \cdots \\ \vdots & \vdots & & \ddots & \ddots \end{pmatrix}.$$

We denote the non-zero blocks of U as A_i^* rather than U_i since we shall see later that these blocks are formed using the A_i blocks of P. Once the matrix G has been formed then L is known. Observe that the inverse of L can be written in terms of the powers of G. For example,

$$\begin{pmatrix} I & 0 & 0 & 0 & \cdots \\ -G & I & 0 & 0 & \cdots \\ 0 & -G & I & 0 & \cdots \\ 0 & 0 & -G & I & \cdots \\ \vdots & \vdots & & \ddots & \ddots \end{pmatrix} \begin{pmatrix} I & 0 & 0 & 0 & \cdots \\ G & I & 0 & 0 & \cdots \\ G^2 & G & I & 0 & \cdots \\ G^3 & G^2 & G & I & \cdots \\ \vdots & \vdots & & \ddots & \ddots \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & \cdots \\ 0 & 1 & 0 & 0 & \cdots \\ 0 & 0 & 1 & 0 & \cdots \\ 0 & 0 & 0 & 1 & \cdots \\ \vdots & \vdots & \vdots & & \ddots \end{pmatrix}.$$

From Equation (18), we have

$$(\pi_0, \pi_1, \cdots, \pi_j, \cdots) \left(\begin{array}{c|ccccc} I - B_{00} & -B_{01} & -B_{02} & -B_{03} & \cdots & -B_{0j} & \cdots \\ \hline -B_{10} & & & & & & \\ 0 & & & & & & \\ 0 & & & UL & & & \\ 0 & & & & & & \\ \vdots & & & & & & \end{array}\right) = (0, 0, \cdots 0, \cdots)$$

which allows us to write

$$\pi_0\,(-B_{01},\ -B_{02},\ \cdots)+(\pi_1,\ \pi_2,\ \cdots)\,UL=0$$

or

$$\pi_0\,(B_{01},\ B_{02},\ \cdots)\,L^{-1}=(\pi_1,\ \pi_2,\ \cdots)\,U,$$

i.e.,

$$\pi_0\,(B_{01},\ B_{02},\ \cdots)\begin{pmatrix} I & 0 & 0 & 0 & \cdots\\ G & I & 0 & 0 & \cdots\\ G^2 & G & I & 0 & \cdots\\ G^3 & G^2 & G & I & \cdots\\ \vdots & \vdots & & \ddots & \ddots \end{pmatrix}=(\pi_1,\ \pi_2,\ \cdots)\,U$$

Forming the product of $(B_{01},\ B_{02},\ \cdots)$ and L^{-1} leads to the important result

$$\pi_0\,(B^*_{01},\ B^*_{02},\ \cdots)=(\pi_1,\ \pi_2,\ \cdots)\,U \tag{19}$$

where

$$\begin{aligned} B^*_{01}&=B_{01}+B_{02}G+B_{03}G^2+\cdots=\sum_{k=1}^{\infty}B_{0k}G^{k-1}\\ B^*_{02}&=B_{02}+B_{03}G+B_{04}G^2+\cdots=\sum_{k=2}^{\infty}B_{0k}G^{k-2}\\ &\vdots\\ B^*_{0i}&=B_{0i}+B_{0,i+1}G+B_{0,i+2}G^2+\cdots=\sum_{k=i}^{\infty}B_{0k}G^{k-i} \end{aligned}$$

The system of equations (19) will allow us to compute the successive components of the vector π once the initial component π_0 and the matrix U are known. To see this, write Equation (19) as

$$\pi_0\,(B^*_{01},\ B^*_{02},\ \cdots)=(\pi_1,\ \pi_2,\ \cdots)\begin{pmatrix} A^*_1 & A^*_2 & A^*_3 & A^*_4 & \cdots\\ 0 & A^*_1 & A^*_2 & A^*_3 & \cdots\\ 0 & 0 & A^*_1 & A^*_2 & \cdots\\ 0 & 0 & 0 & A^*_1 & \cdots\\ \vdots & \vdots & \vdots & & \ddots \end{pmatrix}$$

and observe that

$$\begin{aligned} \pi_0B^*_{01}=\pi_1A^*_1&\Longrightarrow \pi_1=\pi_0B^*_{01}A^{*-1}_1\\ \pi_0B^*_{02}=\pi_1A^*_2+\pi_2A^*_1&\Longrightarrow \pi_2=\pi_0B^*_{02}A^{*-1}_1-\pi_1A^*_2A^{*-1}_1\\ \pi_0B^*_{03}=\pi_1A^*_3+\pi_2A^*_2+\pi_3A^*_1&\Longrightarrow \pi_3=\pi_0B^*_{03}A^{*-1}_1-\pi_1A^*_3A^{*-1}_1-\pi_2A^*_2A^{*-1}_1\\ &\vdots \end{aligned}$$

In general, we find

$$\pi_i = \left(\pi_0 B_{0i}^* - \pi_1 A_i^* - \pi_2 A_{i-1}^* - \cdots - \pi_{i-1} A_2^*\right) A_1^{*-1},\ i = 1, 2, \ldots = \left(\pi_0 B_{0i}^* - \sum_{k=1}^{i-1} \pi_k A_{i-k+1}^*\right) A_1^{*-1}.$$

To compute the first subvector π_0 we return to

$$\pi_0 \left(B_{01}^*,\ B_{02}^*,\ \cdots\right) = \left(\pi_1,\ \pi_2,\ \cdots\right) U$$

and write it as

$$(\pi_0, \pi_1, \cdots, \pi_j, \cdots) \left(\begin{array}{c|cccccc} I - B_{00} & -B_{01}^* & -B_{02}^* & -B_{03}^* & \cdots & -B_{0j}^* & \cdots \\ \hline -B_{10} & A_1^* & A_2^* & A_3^* & \cdots & A_j^* & \cdots \\ 0 & 0 & A_1^* & A_2^* & \cdots & A_{j-1} & \cdots \\ 0 & 0 & 0 & A_1^* & \cdots & A_{j-2}^* & \cdots \\ 0 & 0 & 0 & 0 & & \vdots & \cdots \\ \vdots & & & & & & \end{array}\right) = (0, 0, \cdots 0, \cdots)$$

From the first two equations, we have

$$\pi_0 \left(I - B_{00}\right) - \pi_1 B_{10} = 0$$

and

$$-\pi_0 B_{01}^* + \pi_1 A_1^* = 0.$$

This latter gives

$$\pi_1 = \pi_0 B_{01}^* A_1^{*-1}$$

which when substituted into the first gives

$$\pi_0 \left(I - B_{00}\right) - \pi_0 B_{01}^* A_1^{*-1} B_{10} = 0$$

or

$$\pi_0 \left(I - B_{00} - B_{01}^* A_1^{*-1} B_{10}\right) = 0$$

from which we may now compute π_0, but correct only to a multiplicative constant. It must be normalized so that $\sum_{i=0}^{\infty} \pi_i = 1$ which may be accomplished by enforcing the condition

$$\pi_0 e + \pi_0 \left(\sum_{i=1}^{\infty} B_{0i}^*\right) \left(\sum_{i=1}^{\infty} A_i^*\right)^{-1} e = 1. \tag{20}$$

We now turn our attention to the computation of the matrix U. Since

$$UL = \begin{pmatrix} I - A_1 & -A_2 & -A_3 & \cdots & -A_j & \cdots \\ -A_0 & I - A_1 & -A_2 & \cdots & -A_{j-1} & \cdots \\ 0 & -A_0 & I - A_1 & \cdots & -A_{j-2} & \cdots \\ 0 & 0 & -A_0 & \cdots & -A_{j-3} & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{pmatrix},$$

we have

$$\begin{pmatrix} A_1^* & A_2^* & A_3^* & A_4^* & \cdots \\ 0 & A_1^* & A_2^* & A_3^* & \cdots \\ 0 & 0 & A_1^* & A_2^* & \cdots \\ 0 & 0 & 0 & A_1^* & \cdots \\ \vdots & \vdots & \vdots & & \ddots \end{pmatrix} =$$

$$\begin{pmatrix} I-A_1 & -A_2 & -A_3 & \cdots & -A_j & \cdots \\ -A_0 & I-A_1 & -A_2 & \cdots & -A_{j-1} & \cdots \\ 0 & -A_0 & I-A_1 & \cdots & -A_{j-2} & \cdots \\ 0 & 0 & -A_0 & \cdots & -A_{j-3} & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{pmatrix} \begin{pmatrix} I & 0 & 0 & 0 & \cdots \\ G & I & 0 & 0 & \cdots \\ G^2 & G & I & 0 & \cdots \\ G^3 & G^2 & G & I & \cdots \\ \vdots & \vdots & & \ddots & \ddots \end{pmatrix}$$

and it is now apparent that

$$\begin{aligned} A_1^* &= I - A_1 - A_2G - A_3G^2 - A_4G^3 - \cdots = I - \sum_{k=1}^{\infty} A_kG^{k-1} \\ A_2^* &= -A_2 - A_3G - A_4G^2 - A_5G^3 - \cdots = -\sum_{k=2}^{\infty} A_kG^{k-2} \\ A_3^* &= -A_3 - A_4G - A_5G^2 - A_6G^3 - \cdots = -\sum_{k=3}^{\infty} A_kG^{k-3} \\ &\vdots \\ A_i^* &= -A_i - A_{i+1}G - A_{i+2}G^2 - A_{i+3}G^3 - \cdots = -\sum_{k=i}^{\infty} A_kG^{k-i}, \quad i \geq 2. \end{aligned}$$

We now have all the results we need. The basic algorithm is

- Construct the matrix G.
- Obtain π_0 by solving the system of equations $\pi_0 \left(I - B_{00} - B_{01}^* A_1^{*-1} B_{10}\right) = 0$, subject to the normalizing condition, Equation (20).
- Compute π_1 from $\pi_1 = \pi_0 B_{01}^* A_1^{*-1}$.
- Find all other required π_i from $\pi_i = \left(\pi_0 B_{0i}^* - \sum_{k=1}^{i-1} \pi_k A_{i-k+1}^*\right) A_1^{*-1}$.

where

$$B_{0i}^* = \sum_{k=i}^{\infty} B_{0k}G^{k-i},\ i \geq 1;\ A_1^* = I - \sum_{k=1}^{\infty} A_kG^{k-1} \text{ and } A_i^* = -\sum_{k=i}^{\infty} A_kG^{k-i}, \quad i \geq 2.$$

This obviously gives rise to a number of computational questions. First is the actual computation of the matrix G. We mentioned previously that that can be obtained from its defining equation by means of the iterative procedure:

$$G_{(0)} = 0; \quad G_{(k+1)} = \sum_{i=0}^{\infty} A_i G_{(k)}^i, \quad k = 0, 1, \ldots$$

However, this is rather slow. Neuts proposed a variant that converges faster, namely,

$$G_{(0)} = 0; \quad G_{(k+1)} = (I - A_1)^{-1} \left(A_0 + \sum_{i=2}^{\infty} A_i G_{(k)}^i \right), \quad k = 0, 1, \ldots$$

Among fixed point iterations such as these, the following suggested by Bini and Meini [8], has the fastest convergence

$$G_{(0)} = 0; \quad G_{(k+1)} = \left(I - \sum_{i=1}^{\infty} A_i G_{(k)}^{i-1} \right)^{-1} A_0, \quad k = 0, 1, \ldots$$

Nevertheless, fixed point iterations can be very slow in certain instances. More advanced techniques based on cyclic reduction have been developed and converge much faster. These however, are beyond the realm of this paper.

The second major problem is the computation of the infinite summations that appear in the formulae. Frequently the structure of the matrix is such that A_k and B_k are zero for relatively small (single digit integer) values of k, so that forming these summations in not too onerous. In all cases, the fact that $\sum_{k=0}^{\infty} A_k$ and $\sum_{k=0}^{\infty} B_k$ are stochastic implies that the matrices A_k and B_k are negligibly small for large values of i and can be set to zero once k exceeds some threshold k_M. In this case, we take $\sum_{k=0}^{k_M} A_k$ and $\sum_{k=0}^{k_M} B_k$ to be stochastic. More precisely, a nonnegative matrix P is said to be *numerically stochastic* if $Pe < \mu e$ where μ is the precision of the computer. When k_M is not small, finite summations of the type $\sum_{k=i}^{k_M} Z_k G^{k-i}$ should be evaluated using Horner's rule. For example, if $k_M = 5$

$$Z_1^* = \sum_{k=1}^{5} Z_k G^{k-1} = Z_1 G^4 + Z_2 G^3 + Z_3 G^2 + Z_4 G + A_5$$

should be evaluated from the inner-most parenthesis outwards as

$$Z_1^* = (\, [\, (Z_1 G + Z_2) G + Z_3 \,] G + Z_4 \,) \, G + Z_5.$$

4 The Future

Today, there are many many researchers who work in the field of numerical analysis, possibly many times more than those who research interests lie in performance evaluation. The performance evaluation community has been slow to venture into the numerical analysis arena, probably because there has not been a great need to. The success that has accompanied queueing modelling has largely eliminated the need to set up and solve global balance equations numerically. However, as models become more complex, it is becoming increasing evident that there is place for numerical analysis methods in the modelers toolbox. The challenges for the future will be for performance analysts to keep abreast of the research conducted by numerical analysts in the areas of aerospace, civil engineering, and so on and to successfully apply these novel approaches to the special structures that arise in performance evaluation.

References

[1] M. Ajmone Marsan, G. Balba and G. Conte. A class of generalized stochastic Petri nets for the performance evaluation of multiprocessor systems. *ACM Trans. Comput. Systems*, Vol. 2, No. 2, pp 93–122, 1994.

[2] G.P. Basharin and V.A. Naoumov. The life and work of A.A. Markov. *Proceedings for the Fourth International Conference on the Numerical Solution of Markov Chains*, Urbana, Champaign. A.N. Langville and W.J. Stewart, Eds., pp. 1–10, 2003.

[3] S.C. Allmaier and G. Horton. Parallel shared-memory state-space exploration in stochastic modelling. Lecture Notes in Compuer Science, #1253. Springer 1997.

[4] A. Benoit, B. Plateau and W.J. Stewart. Memory-efficient Kronecker algorithms with applications to modelling parallel systems. *Future Generation Computer Systems: (FGCS) Special Issue on "System Performance Analysis and Evaluation* Volume 22, Issue 7, pages 838–847, August 2006.

[5] M. Benzi. A direct projection method for Markov chains. *Proceedings for the Fourth International Conference on the Numerical Solution of Markov Chains*, Urbana, Champaign. A.N. Langville and W.J. Stewart, Eds., pp. 11–30, 2003.

[6] S. Berson, E. de Souza e Silva and R.R. Muntz. A methodology for the specification and generation of Markov models. *Numerical Solution of Markov Chains*, William J. Stewart, Ed., Marcel Dekker Inc., New York, pp. 11–36, 1991.

[7] S. Berson and R.R. Muntz. Detecting block GI/M/1 and block M/G/1 matrices from model specifications. *Computations with Markov Chains*, W.J. Stewart, Ed., Kluwer International Publishers, Boston, pp. 1–19, 1995.

[8] D. Bini and B. Meini. On cyclic reduction applied to a class of Toeplitz matrices arising in queueing problems. In W.J. Stewart, Editor, *Computations with Markov Chains,* pp. 21–38, Kluwer Academic Publishers, Boston, MA, 1995.

[9] P. Buchholz and T. Dayar. Block SOR for Kronecker structured representations. *Proceedings for the Fourth International Conference on the Numerical Solution of Markov Chains*, Urbana, Champaign. A.N. Langville and W.J. Stewart, Eds., pp. 121–144, 2003.

[10] P. Buchholz. Equivalence relations for stochastic automata networks. *Computations with Markov Chains*, W.J. Stewart, Ed., Kluwer International Publishers, Boston, pp. 197–215, 1995.

[11] P. Buchholz. An aggregation-disaggregation algorithm for stochastic automata networks. *Prob. in the Eng and Inf. Sci.*, Vol. 11, pp. 229–254, 1997.

[12] S. Caselli, G. Conte and P. Marenzoni. Parallel state space exploration for GSPN models. Lecture Notes in Computer Science # 935. Proceedings of the 16th International Conference on the Application and Theory of Petri Nets. Springer Verlag, Turin, Italy, June 1995.

[13] W-L. Cao and W.J. Stewart. Iterative aggregation/disaggregation techniques for nearly uncoupled Markov chains. *Journal of the ACM*, Vol 32, pp. 702–719, 1985.

[14] *Advances in matrix-analytic methods for stochastic models,* S.R. Chakravarthy and A.S. Alfa, Ed. Lecture Notes in Pure and Applied Mathematics, 1998.

[15] G. Ciardo, J. Muppala and K.S. Trivedi. SPNP: stochastic Petri net package. In *Proceedings of the Third International Workshop on Petri Nets and Performance Models*, (PNPM89), Kyoto, Japan, 1989.

[16] G. Ciardo and M. Tilgner. On the use of Kronecker operators for the solution of generalized stochastic Petri nets. ICASE Report 96–35, Hampton, VA, may 1996.

[17] G. Ciardo and A.S. Miner. Storage alternatives for large structured state spaces. *IEEE International Computer Performance and Dependability Symposium*, R. Marie et al., eds., Springer Verlag, LNCS 1245, pp. 44–57, 1997.

[18] P.J. Courtois. *Decomposability*. Academic Press, New York, 1977.

[19] C. A. Clarotti. The Markov approach to calculating system reliability : Computational Problems. In A. Serra and R. E. Barlow, editors, *International School of Physics "Enrico Fermi", Varenna, Italy*, pp. 54–66, Amsterdam, 1984. North-Holland Physics Publishing.

[20] J.N. Daigle and D.M. Lucantoni. Queueing systems having phase-dependent arrival and service rates. *Numerical Solution of Markov Chains*, William J. Stewart, Ed., Marcel Dekker Inc., New York, pp. 161–202, 1991.

[21] S. Donatelli. Superposed stochastic automata: A class of stochastic Petri nets with parallel solution and distributed state space. *Performance Evaluation*, Vol. 18, pp. 21–36, 1993.

[22] A.I. Elwalid, D. Mitra and T.E. Stern. Theory of statistical multiplexing of Markovian sources: Spectral expansions and algorithms. *Numerical Solution of Markov Chains*, William J. Stewart, Ed., Marcel Dekker Inc., New York, pp. 223–238, 1991.

[23] P. Fernandes, B. Plateau and W.J. Stewart. Efficient descriptor-vector multiplication in stochastic automata networks. *Journal of the ACM*, Vol. 45, No. 3, May 1998.

[24] J-M Fourneau, M. Lecoz and F. Quessette. Algorithms for an irreducible and lumpable strong stochastic bound. *Proceedings for the Fourth International Conference on the Numerical Solution of Markov Chains*, Urbana, Champaign. A.N. Langville and W.J. Stewart, Eds., pp. 191–206, 2003.

[25] J-M. Fourneau and F. Quessette. Graphs and stochastic automata networks. *Computations with Markov Chains*. W.J. Stewart, Ed., Kluwer International Publishers, Boston, 1995.

[26] G. Franceschinis and R. Muntz. Computing bounds for the performance indices of quasi-lumpable stochastic well-formed nets. *Proceedings of the 5th International Workshop on Petri Nets and Performance Models*, Toulouse, France, IEEE Press, pp. 148–157, October 1993.

[27] S.B. Gershwin, I.C. and Schick. Modelling and analysis of three-stage transfer lines with unreliable machines and finite buffers. *Operations Research*, 31, 2, pp. 354–380, 1983.

[28] G. H. Golub and C. F. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, Baltimore, second edition, 1989.

[29] W. Grassmann. Transient solutions in Markovian queueing systems. *Comput. Opns. Res.*, 4:47–56, 1977.

[30] W. Grassmann, M.I. Taksar and D.P. Heyman. Regenerative analysis and steady state distributions, *Operations Research*, Vol. 33, pp 1107–1116, 1985.

[31] D. Gross and D. R. Miller. The randomization technique as modelling tool and solution procedure for transient Markov processes. *Operations Research*, 32(2):343–361, 1984.

[32] J. Hillston. Computational Markovian modelling using a process algebra. *Computations with Markov Chains*, W.J. Stewart, Ed., Kluwer International Publishers, Boston, 1995.

[33] M.A. Jafari and J.G. Shanthikumar. Finite state spacially non-homogeneous quasi birth-death processes. Working Paper #85-009, Dept. of Industrial Engineering and Operations Research, Syracuse University, Syracuse, New York 13210.

[34] P. Kemper. Closing the gap between classical and tensor based iteration techniques. *Computations with Markov Chains*, W.J. Stewart, Ed., Kluwer International Publishers, Boston, 1995.

[35] R. Klevans and W.J. Stewart. From queueing networks to Markov chains: The XMarca interface. *Performance Evaluation*, Vol. 24, pp. 23–45, 1995.

[36] U. Krieger, B. Muller-Clostermann and M. Sczittnick. Modeling and analysis of communication systems based on computational methods for Markov chains. *IEEE Journ. on Selec. Ar. in Comm*, Vol. 8, pp. 1630–1648, 1990.

[37] W.J. Knottenbelt. *Generalized Markovian analysis of timed transition systems.* Master's thesis, University of Capetown, 1995.

[38] W.J. Knottenbelt, M. Mestern, P. Harrison and P. Kritzinger. Probability, parallelism and the state space exploration problem. Lecture Notes in Computer Science, #1469. R. Puigjaner, N.N. Savino and B. Serra, Eds. Springer Publishers, 1998.

[39] G. Latouche and Y. Ramaswami. A logarithmic reduction algorithm for quasi birth and death processes. Technical Report, Bellcore TM-TSV-021374 and Université Libre de Bruxelles, Sémin. Théor. Prob., Rapp. Tech. 92/3, 1992.

[40] S.Q. Li and J.W. Mark. Performance of voice/data integration on a TDM system. *IEEE Trans. Communications*, Vol. COM-33, No. 12, Dec. 1985, pp. 1265–1273.

[41] C. Lindemann. Exploiting isomorphisms and special structures in the analysis of Markov regenerative stochastic Petri nets. *Computations with Markov Chains*, W.J. Stewart, Ed., Kluwer International Publishers, Boston, pp. 383–402, 1995.

[42] M. Malhotra and K. S. Trivedi. Higher-order methods for transient analysis of stiff Markov chains. Duke University Technical Report DUKE-CCSR-91, Center for Computer Systems Research, Durham, USA, 1991.

[43] I. Marek and D.B. Szyld. Algebraic Schwartz methods for the numerical solution of Markov chains. *Proceedings for the Fourth International Conference on the Numerical Solution of Markov Chains*, Urbana, Champaign. A.N. Langville and W.J. Stewart, Eds., pp. 45–56, 2003.

[44] D.F. McAllister, G.W. Stewart and W.J. Stewart. On a Rayleigh-Ritz refinement technique for nearly uncoupled stochastic matrices. *Journal of Linear Algebra and Its Applications,* Vol. 60, pp. 1–25, August 1984.

[45] C.D. Meyer. Stochastic complementation, uncoupling Markov chains and the theory of nearly reducible systems. *SIAM Rev.*, Vol. 31, pp. 240–272, 1989.

[46] B. Meini. *New convergence results on functional techniques for the numerical solution of M/G/1 type Markov chains.* em Numer. Math,, Vol. 78, pp. 39–58, 1997.

[47] C. B. Moler and C. F. Van Loan. Nineteen dubious ways to compute the exponential of a matrix. *SIAM Review.*, Vol. 20, No. 4, pp. 801–836, October 1978.

[48] E.J. Muth and S. Yeralan. Effect of buffer size on productivity of work stations that are subject to breakdowns. The 20-th IEEE Conference on Decision and Control, pp. 643–648, 1981.

[49] M.F. Neuts. *Matrix Geometric Solutions in Stochastic Models – An Algorithmic Approach*. The Johns Hopkins University Press, Baltimore, MD, 1981.

[50] M.F. Neuts. *Structured Stochastic Matrices of M/G/1 Type and Their Applications.* Marcel Dekker, New York, 1989.

[51] B. Philippe, Y. Saad, and W. J. Stewart. Numerical methods in Markov chains modeling. *Operations Research*, Vol. 40, No. 6, 1992.

[52] B. Philippe and R. B. Sidje. Transient solutions of Markov processes by Krylov subspaces. *Computations with Markov Chains*, W.J. Stewart, Ed., Kluwer International Publishers, Boston, pp. 95–119, 1995.

[53] B. Plateau. On the stochastic structure of parallelism and synchronization models for distributed algorithms. *Proc. ACM Sigmetrics Conference on Measurement and Modelling of Computer Systems*, Austin, Texas, August 1985.

[54] B. Plateau and K. Atif. Stochastic automata network for modelling parallel systems. *IEEE Trans. on Software Engineering*, Vol. 17, No. 10, pp. 1093–1108, 1991.

[55] B. Plateau, J.M. Fourneau and K.H. Lee. PEPS: A package for solving complex Markov models of parallel systems. In R. Puigjaner, D. Potier, Eds., *Modelling Techniques and Tools for Computer Performance Evaluation*, Spain, September 1988.

[56] M. Veran and D. Potier. QNAP2: A portable environment for queueing systems modelling. In D. Potier, ed. *Modelling and Tools for Performance Analysis*, pp. 25–63. Elsevier Science Publishers, 1985.

[57] V. Ramaswami. A stable recursion for the steady state vector in Markov chains of M/G/1 type. *Commun. Statist. Stochastic Models*, Vol. 4, pp. 183–188, 1988.

[58] V. Ramaswami. Nonlinear matrix equations in applied probability — Solution techniques and open problems. *SIAM Review*, Vol.30, No. 2, pp. 256–263, June 1988.

[59] V. Ramaswami and M.F. Neuts. Some explicit formulas and computational methods for infinite server queues with phase type arrivals. *J. Appl. Probab.*, Vol. 17, pp. 498–514, 1980.

[60] V. Ramaswami and D.M. Lucantoni. Stationary waiting time distributions in queues with phase-type service and in quasi-birth-and-death processes. *Comm. Statist. Stochastic Models*, Vol. 1, pp. 125–136, 1985.

[61] A. Reibman and K. Trivedi. Transient analysis of cumulative measures of Markov model behavior. *Commu. Statist.-Stochastic Models*, 5(4):683–710, 1989.

[62] Y. Saad. *Iterative solution of sparse linear systems.* PWS Publishing, New York, 1996.

[63] Y. Saad. *Numerical methods for large eigenvalue problems*. John Wiley & Sons, Manchester Univ. Press, 1992.

[64] I. Sbeity, B. Plateau, L. Brenner and W.J. Stewart. Phase-Type Distributions in Stochastic Automata Networks. To appear in *European Journal of Operations Research*.

[65] P.J. Schweitzer. Aggregation methods for large Markov chains. In *Mathematical Computer Performance and Reliability*, G. Iazeolla, P.J. Courtois and A. Hordijk, Eds. North-Holland, Amsterdam, pp 275–286, 1984.

[66] P. Semal. Two bounding schemes for the steady state solution of Markov chains. *Computations with Markov Chains*, W.J. Stewart, Ed., Kluwer International Publishers, Boston, pp. 307–320, 1995.

[67] E. Seneta. MArkov and the Creation of Markov Chains. *MAM-2006, MArkov Anniversary Meeting: An International Conference to Celebrate the 150th Anniversary of the Birth of A.A. Markov. Boson Books, Raleigh, N. Carolina.* pp. 1 – 20, 2006.

[68] R. B. Sidje. *Parallel algorithms for large sparse matrix exponentials: Application to numerical transient analysis of Markov processes*. PhD thesis, University of Rennes 1, July 1994.

[69] R. B. Sidje. EXPOKIT. A software package for computing matrix exponentials. Accepted for publication in ACM-Transactions of Mathematical Software, 1997.

[70] M. Squillante. MAGIC: A computer performance modeling tool based on matrix geometric techniques. *Computer Performance Evaluation: Modeling Techniques and Tools*, G. Balbo and G. Serazzi, Eds. North-Holland Publishers, pp. 411–425, 1992.

[71] W.J. Stewart. *An Introduction to the Numerical Solution of Markov Chains*, Princeton University Press, New Jersey, 1994.

[72] W.J. Stewart, K. Atif and B. Plateau. The numerical solution of stochastic automata networks. *European Journal of Operations Research*, Vol. 86, No. 3, pp. 503–525, 1995.

[73] W.J. Stewart. MARCA: Markov chain analyzer. *IEEE Computer Repository* No. R76 232, 1976. (See the URL: *http://www.csc.ncsu.edu/faculty/WStewart/*)

[74] School of Mathematics and Statistics University of St Andrews, Scotland December 1996 URL: *http://www-history.mcs.st-andrews.ac.uk/history/Mathematicians/Markov.html*

[75] J-M Vincent and C. Marchand. On the exact simulation of functional of stationary Markov chains. *Proceedings for the Fourth International Conference on the Numerical Solution of Markov Chains*, Urbana, Champaign. A.N. Langville and W.J. Stewart, Eds., pp. 77–98, 2003.

[76] P. Von Hilgers and A.N. Langville. The five greatest applications of Markov Chains. *MAM-2006, MArkov Anniversary Meeting: An International Conference to Celebrate the 150th Anniversary of the Birth of A.A. Markov. Boson Books, Raleigh, N. Carolina.* pp. 155 – 168, 2006.

[77] V.I. Wallace and R.S. Rosenberg. RQA-1, The recursive queue analyzer. Technical report No.2, Systems Engineering Laboratory, University of Michigan, Ann Arbor, Feb. 1966.

[78] V. Wallace. The solution of quasi birth and death processes arising from multiple access computer systems. Ph.D. Dissertation, Systems Engineering Laboratory, University of Michigan, Tech. Report No. 07742-6-T, 1969.

[79] Wong, Giffin, and Disney, Two finite M/M/1 queues in tandem: A matrix solution for the steady state. *OPSEARCH* 14, 1, pp 1–18, 1977.

[80] T. Yang, M.J.M. Posner and J.G.C. Templeton. A generalized recursive technique for finite Markov processes. *Numerical Solution of Markov Chains*, William J. Stewart, Ed., Marcel Dekker Inc., New York, pp. 203–221, 1991.

[81] J. Ye and S.Q. Li. Analysis of multi-media traffic queues with finite buffer and overload control — Part I: Algorithms. *Proc. of INFOCOM '91*, pp. 1464–1474.

[82] J. Ye and S.Q. Li. Analysis of multi-media traffic queues with finite buffer and overload control — Part II: Applications. *Proc. of INFOCOM '92*, pp. 848–859.

Queueing Networks

Simonetta Balsamo and Andrea Marin

Dipartimento di Informatica
Università Ca' Foscari di Venezia
Via Torino 155, 30172 Venezia Mestre, Italy
{balsamo,marin}@dsi.unive.it

Abstract. The purpose of this tutorial is to survey queueing networks, a class of stochastic models extensively applied to represent and analyze resource sharing systems such as communication and computer systems. Queueing networks (QNs) have been proved to be a powerful and versatile tool for system performance evaluation and prediction. First we briefly survey QNs that consist of a single service center, i.e., the basic queueing systems defined under various hypotheses, and we discuss their analysis to evaluate a set of performance indices, such as resource utilization and throughput and customer response time. Their solution is based on the introduction of an underlying stochastic Markov process. Then, we introduce QNs that consist of a set of service centers representing the system resources that provide service to a collection of customers that represent the users. Various types of customers define the customers classes in the network that are gathered in chains. We consider various analytical methods to analyze QNs with single-class and multiple-class. We mostly focus on product-form QNs that have a simple closed form expression of the stationary state distribution that allows to define efficient algorithms to evaluate average performance measures. We review the basic results, stating from the BCMP theorem that defines a large class of product-form QNs, and we present the main solution algorithms for single-class e multiple-class QNs. We discuss some interesting properties of QNs including the arrival theorem, exact aggregation and insensitivity. Finally, we discuss some particular models of product-form QNs that allow to represent special system features such as state-dependent routing, negative customers, customers batch arrivals and departures and finite capacity queues. The class of QN models is illustrated through some application examples of to analyze computer and communication systems.

1 Introduction

Queueing network models have been extensively applied to represent and analyze resource sharing systems, such as production, communication and computer systems. They have proved to be a powerful and versatile tool for system performance evaluation and prediction. A queueing network model (QN) is a collection of service centers representing the system resources that provide service to a collection of customers that represent the users. Customers compete for the resource

M. Bernardo and J. Hillston (Eds.): SFM 2007, LNCS 4486, pp. 34–82, 2007.

service and they possibly wait to be served in the queue into the service centers, according to the queueing discipline. The analysis of QNs consists of evaluating a set of performance measures, such as resource utilization, throughput and customer response time. The dynamic behavior of a QN can be described by a set of random variables that define a stochastic process. Under some constraints on the QN it is possible to define an associated underlying stochastic Markov process, and to compute the QN performance indices by its solution.

The popularity of QNs for system performance evaluation is due to a good balance between a relative high accuracy in the performance results and the efficiency in model analysis and evaluation. In this framework the class of product form networks has played a fundamental role. Product-form QNs have a simple closed form expression of the stationary state distribution that allow us to define efficient algorithms to evaluate average performance measures with polynomial time complexity in the number of model components.

In this work we introduce QN models and their properties. QNs extend the basic queueing systems that are stochastic models first introduced to represent the entire system by a single service center. Queueing systems have been first applied to analyze congestion in telephonic systems and then to study congestion in computer and communication systems [42,32,48,76,49,39]. A QN represents a congestion and resource sharing systems as a network of interacting service centers whose analysis often provides quite accurate prediction of their performance. Despite of several assumptions of the class of queueing networks, they have been observed to be very robust models [74]. QNs can be analyzed by analytical methods or by simulation. Simulation is a general technique of wide application, but its main drawback is the potential high development and computational cost to obtain accurate results. Analytical methods require that the model satisfies a set of assumptions and constraints and are based on a set of mathematical relationships that characterize the system behavior.

We consider analytical methods to analyze QNs and we mostly focus on product-form QNs that have a simple closed-form of the stationary state probability distribution, which allow the definition of efficient algorithms to evaluate their performance. Jackson [38] introduced product-form QNs for open exponential networks and Gordon and Newell [33] for closed exponential networks. They introduce several assumptions on the model characteristics and provide a simple closed-form expression of the stationary state distribution and some average performance indices. This class of models was then extended to include several interesting and useful characteristics to represent more complex system. These features include different types of customers of the networks, various queueing discipline (i.e., the scheduling algorithms of the waiting queues), state-dependent service rate, state-dependent routing between the service centers and some constraints on the population of subnetworks. The main result concerning product-form QNs known as the BCMP theorem was presented by Baskett, Chandy, Muntz and Palacios [9]. It defines the well-known class of BCMP QNs with product-form solution for open, closed or mixed models with multiple classes of customers, various service disciplines and service time

distributions and some types of load-dependent functions for the arrival process and the customers service time. The stationary state distribution is expressed as the product of the distributions of the single service centers with appropriate parameters and, for closed networks, with a normalization constant. An important property of product-form QNs is the arrival theorem. It states that the distribution at arrival times at a service center is identical to the distribution at arbitrary times of the same network, for open networks, and of a network with one less customer for closed networks [47,70]. This led to the definition of a set of recurrence equations between average performance measure for closed networks, and hence to a recursive computational algorithm, the Mean Value Analysis (MVA) [64], that avoids the direct evaluation of the normalization constant.

Various computational algorithms can be applied to analyze and to evaluate the performance indices of product-form QNs. The relevance of these solution algorithms is twofold. First, they provide a powerful tool in the efficient analysis of large QN models, and the analyst can choose the appropriate and most convenient algorithm depending on the type of model. Second, these algorithms provides a basis for approximate solution methods of more general network model with and without product-form. The most relevant solution algorithms for closed networks are the Convolution Algorithm [13] and the Mean Value Analysis [64]. They provide the evaluation of average performance indices with a polynomial space and time computational complexity in the network dimension that is the number of service centers and the network population. Product-form networks with multiple classes of customers are more difficult to analyze. Various types of customers define the customer classes in the network that are gathered in chains. Both Convolution and MVA algorithms have been extended to multiple-class networks [62,67,45,12], but their cost grows exponentially with the number of customer classes or chains. Other algorithms for multiple-class QNs have been proposed. The tree Convolution and tree MVA algorithms for multiple-chain networks are based on a tree data structure to optimize the algorithm computation [46,77,37]. Other algorithms for multiple-chain QNs with several types of customers are Recursion by Chain Algorithm (RECAL) [23,24], Mean Value Analysis by Chain [22] and Distribution Analysis by Chain (DAC) [26]. Their computational complexity is polynomial with the number of classes of customers, but exponential in the number of service centers.

The main computational algorithms for QNs have been integrated in various software tools for performance modelling and analysis that include user friendly interfaces based on different languages to take into account the particular field of application, e.g., computer networks, computer systems. This allows not expert users to apply efficient performance modelling techniques [49,66,76,14]. More recently the solution performance algorithms for QNs have been integrated with model specification techniques to provide tools for the combined functional and quantitative system analysis [72].

Product-form networks yield various interesting properties. The insensitivity property states that the analytical results, i.e., the stationary state distribution and the average performance indices, depend on the service time requirements

only through their average. Similarly, the performance indices depend on the customers routing only through the average visit ratio to each service center [9,17,18,71,79]. Another important property of product-form QN models is that aggregation methods yield exact results. The aggregation theorem, introduced by Chandy, et al. [15], allows substituting a subnetwork with a single service center, so that the new aggregated network has the same behavior of the original one in terms of a set of performance indices. From the performance viewpoint exact aggregation allows us to apply the hierarchical system design process by relating the performance indices of the models at different levels in the hierarchy [49]. In a bottom-up analysis of systems represented by a succession of QNs exact aggregation defines the next model. Similarly, in a hierarchical top-down design of system with given performance requirements, the inverse process of disaggregation or development of the network can be applied to define a more detailed model with the same performance indices [7]. Aggregation is an efficient technique when applied to the analysis of nearly complete decomposable systems. Informally, such a system can be decomposed into subsystems whose internal interactions are much higher than the interactions among the subsystems [25]. Exact aggregation for product-form QNs provides a basis for approximate solution methods of more general non-product form network models [51].

More recently further research has devoted to the extension of the class of product form network models and to its characterization. Some interesting new features have been defined such as the G-Networks, that are QNs with positive and negative customers proposed by Gelenbe [29] that can be used to represent special dynamic of actual systems. Moreover G-Networks can be seen as a unifying model for neural nets and queueing networks [30]. Some other more complex models include various functions of state-dependent routing and several special cases of QNs with finite capacity queues, finite population constraints and blocking [2,5,11,34,44,75,78]. Nelson in [56] discusses the mathematics leading to the product-form results and the properties of the stochastic process underlying the network model. Some extensions of product-form QN are presented in [78]. Product-form solution has been extended to QNs with batch arrivals and batch services [35,36] that are also related to discrete time QN models.

Some extensions of non product-form QNs have been proposed to represent special classes of systems and communication models, such as Layered Queueing Networks and Extended QN to represent more complex system, e.g., with simultaneous resource possesion, finite capacity queues and blocking, and fork and join [48,68,4,65,61,5].

We shall now provide an introduction to QN models applied to represent and analyze resource sharing systems In particular we consider the class of product-form QNs, the main analytical methods to derive a significant set of performance indices, some relevant QNs properties and their applications to system performance evaluation. We present the basic results, the key ideas and the main algorithms and solution techniques and we discuss the relevance of such models for system performance evaluation.

In Section 2 we introduce the basic queueing systems, that are QNs formed by a single service center, and we recall the definition of the associated continuous-time Markov process. Section 3 presents QNs which consist of a set of interconnected service centers, the definition of multiple-chain and multiple-class models. We briefly review Markovian networks and then we focus on product-form QNs, their characterization, and some interesting extensions and properties. Section 4 deals with the analysis of QNs. We review the most relevant exact analytical algorithms for product-form QNs, which include Convolution and MVA. Then we discuss some approximate solution algorithms for QNs. Some examples of application of QN models for performance analysis of computer and communication systems are presented in Section 5.

2 Queueing Systems

The simplest queueing network consists of a single service center that models the entire system. Basic queueing systems have been defined in queueing theory and applied to analyze congestion systems. The analysis of queueing systems relies on the theory of stochastic processes [21,42,48,68,39,76]. Under appropriate independence and exponential assumptions on the model random variables it is possible to define a continuous-time Markov processes associated to the queueing system. Then queueing system analysis is usually based on the solution of the underlying associated Markov process. We shall now first briefly review Markov process solution that are useful for the analysis of queueing models, and then we introduce some queueing systems. Queueing networks that consist of more than one service centers are introduced in the next section.

2.1 Markovian Stochastic Processes

We shall now review the stochastic Markov process definition and solution that are used to analyze queueing models. A stochastic process is a set of random variables $\{X(t)|t \in T\}$ defined over the same probability space and indexed by the parameter t, called time. The process random variables take values in the set Γ called state space of the process. Both set T and state space Γ can be either discrete or continuous. The process is called continuous-time or discrete-time if the time parameter t is continuous or discrete, respectively. A discrete-time process is usually denoted by $\{X_n|n \in T\}$. If the state space Γ is discrete then the process is called discrete-space or chain, otherwise the process is called continuous-space. The probabilistic behavior of stochastic process is defined by the joint probability distribution function of the random variables $X(t_i)$ for any set of times $t_i \in T, 1 \leq i \leq n, n \geq 1$, denoted by $Pr\{X(t_1) \leq x_1; X(t_2) \leq x_2; \ldots; X(t_n) \leq x_n\}$, where $x_i \in \Gamma$.

A discrete-time process is a Markov process if the state at time $n+1$ only depends on the state probability at time n and is independent of the previous history. This is known as the Markov property. The process conditional probability distribution satisfies the following condition:

$$Pr\{X_{n+1} = j|X_0 = i_0; X_1 = i_1; \ldots; X_n = i_n\} = Pr\{X_{n+1} = j|X_n = i_n\}, \quad (1)$$

for all $n > 0$, and $j, i_0, i_1, \ldots, i_n \in \Gamma$.

Similarly, a continuous-time process is said to be a Markov process if it satisfies the following condition:

$$Pr\{X(t) = j|X(t_0) = i_0; X(t_1) = i_1; \ldots; X(t_n) = i_n\} = Pr\{X(t) = j|X(t_n) = i_n\}, \quad (2)$$

for all set of times $t_0 < t_1 < \ldots < t_n < t$, and $n > 0$, $j, i_0, i_1, \ldots, i_n \in \Gamma$. Note that, because of the Markov property, the residence time of the process in each state is distributed according to either the geometric or the negative exponential distribution respectively for discrete-time or continuous-time Markov processes. Hereafter we consider discrete-space Markov process, also called Markov chain. Consider a discrete-time Markov chain. If the one-step conditional probability on the right-hand side of formula (1) is independent on time n, then the Markov chain is homogeneous. Then we define transition probability from state i to state j as $p_{ij} = Pr\{X_{n+1} = j|X_n = i\}$, and the matrix of state transition probabilities $\mathbf{P} = [p_{ij}]$, where $p_{ij} \in [0, 1]$, $\sum_j p_{ij} = 1$, $\forall i, j \in \Gamma$. The stationary behavior of the Markov process can be evaluated if the process satisfies some conditions. Informally, a Markov process is said to be irreducible if every state can be reached from any other state. Each state can be transient or recurrent, and it is said to be positive recurrent if the average return time to the state is finite. An ergodic Markov chain is irreducible and formed by positively recurrent aperiodic states. Let $\boldsymbol{\pi} = [\pi_0 \pi_1 \pi_2 \ldots]$ denote the stationary state probability vector, where $\pi_j = Pr\{X = j\}$ is the stationary probability of state $j \in \Gamma$. Then for homogeneous ergodic discrete-time Markov chain we can compute the stationary probability $\boldsymbol{\pi}$ as follows [42]:

$$\boldsymbol{\pi} = \boldsymbol{\pi}\mathbf{P}, \quad (3)$$

with the normalizing condition $\sum_j \pi_j = 1$. This is called system of global balance equations.

Let us now consider a continuous-time Markov chain. The Markov chain is homogeneous if the conditioned probability on the right-hand side of formula (2) is independent on time t_n, but only depends on the interval width $(t - t_n)$. In other words we can write the state transition probability from state i to state j only dependent on the interval width s as follows:

$$p_{ij}(s) = Pr\{X(t_n + s) = j|X(t_n) = i\}, \quad \forall i, j \in \Gamma, \forall t_n \geq 0.$$

Hence we have a width dependent state transition probability matrix $\mathbf{P}(s) = [p_{ij}(s)]$. Then we can define a rate transition probability matrix $\mathbf{Q} = [q_{ij}], i, j \in \Gamma$, also called process infinitesimal generator, as follows:

$$\mathbf{Q} = \lim_{s \to 0} \frac{\mathbf{P}(s) - \mathbf{I}}{s}.$$

The stationary behavior of the continuous-time Markov chain can be evaluated for homogeneous ergodic chain. The stationary state probability $\boldsymbol{\pi} = [\pi_0 \pi_1 \pi_2 \ldots]$, where $\pi_j = Pr\{X = j\}$ for each state $j \in \Gamma$, can be computed by solving the following system of global balance equations:

$$\boldsymbol{\pi}\mathbf{Q} = \mathbf{0}, \tag{4}$$

with the normalizing condition $\sum_j \pi_j = 1$.

For the special class of birth-death processes it is possible to derive a closed-form solution of the stationary state probability $\boldsymbol{\pi}$ defined by system (3) for discrete-time and system (4) for continuous-time processes, respectively. A birth-death Markov process has state space $\Gamma = \mathbb{N}$ and the only non-zero state transitions are those from any state i to states $i-1$, i, $i+1$, $\forall i \in \Gamma$. Hence the transition state probability matrix $\mathbf{P}$ for discrete-time, or the transition rate matrix $\mathbf{Q}$ for continuous-time process, is tridiagonal. Let us denote the rates of matrix $\mathbf{Q}$ for a continuous-time birth-death Markov chain as follows: $q_{i\ i+1} = \lambda_i$, $i \geq 0$ and $q_{i\ i-1} = \mu_i$, $i \geq 1$. Then the stationary state probability $\boldsymbol{\pi}$ can be calculated as follows:

$$\pi_i = \pi_0 \prod_{j=0}^{i-1} \frac{\lambda_j}{\mu_{j+1}} \tag{5}$$

for $i \geq 0$, and where π_0 is given by the normalizing condition, i.e.,

$$\pi_0 = \Big[\sum_{i=0}^{\infty} \prod_{j=0}^{i-1} \frac{\lambda_j}{\mu_{j+1}} \Big]^{-1}.$$

This solution holds under the stability condition. A sufficient condition for the stationary solution is that there exists a state $k_0 > 0 : \lambda_k < \mu_k$, $\forall k > k_0$ [42]. Several basic queueing systems can be analyzed by birth-death Markov processes, as we shall now present.

2.2 Basic Queueing Systems

A simple queueing system or service center is illustrated in Figure 1. The system models the flow of customers as they arrive, wait in the queue if the server is busy serving another customer, receive service, and eventually leave the system. For example a uniprocessor computer system can be modeled by a simple queueing system where the program to be executed are the customers, the processes ready for execution are in the queue, the processor is the server whose service models program execution. To describe the behavior of a queueing system in time, we have to specify five basic characteristics: 1) the arrival process, 2) the number of servers, 3) the service process, 4) the service or queueing discipline and 5) the system or queue capacity.

1) The *arrival process* to a queueing system describes the behavior of customers arrivals. We define the interarrival time as a random variable representing the time between two consecutive arrivals. The mean arrival rate is the average

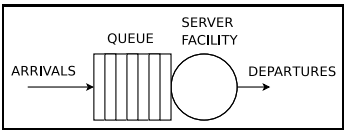


Fig. 1. A queueing system

number of arrivals per unit of time, and is denoted by λ. The Poisson process is often assumed as arrival process. This corresponds to an exponential interarrival distribution.

2) The set of identical parallel *servers* can simultaneously service the customers. Each server may correspond to a physically or logically separate service facility of the system, with a common queue shared by all customers.

3) The *service process* describes the customer service. We define the service time as a random variable representing the time spent for a customer service, whose average is the mean service rate denoted by μ.

4) The *queueing discipline* describes the scheduling algorithm for the customers in the queue. If a customer arrives at the system at a time the server(s) is unavailable to provide service to it, it is forced to wait in the queue temporarily until it can start receiving service. If there are more than one customer waiting in the queue at a time the server becomes available, one of the customers in the queue is selected to start receiving service. The customers' selection for service is referred to as the queueing discipline. Queueing discipline may depend on the arrival time, the customer priority and the possible service already given to the customer. Classical examples of queueing disciplines depending only on the arrival time are First Come First Served (FCFS) and Last Come First Served (LCFS).

5) Finally, the *system capacity* is the upper limit on the number of customers (waiting for and receiving service) in the system. Most analytical studies require the queue size to be infinite, i.e., large enough to accommodate all arriving customers. However, systems have often finite resources imposing an upper bound on the number of customers that can be waiting in the queue simultaneously.

The Kendall's notation A/B/X/Y/Z describes the queueing process of a single queueing system where A indicates the arrival process, B the service process, X the number of parallel servers, Y the system capacity, and Z the service discipline. The simplified notation A/B/X describes a queueing system with infinite capacity and FCFS queueing discipline. Arrival and service processes are denoted by symbols of probability distributions, e.g., D for Deterministic, M for exponential, G for general distribution. For example M/M/1 denotes the system with Poisson (Markov) arrival process, exponential (Markov) service process and a single server and M/G/1 the same system except for the service time that has a general or arbitrary distribution.

The analysis of a queueing system aims to evaluate a set of performance indices, including the following ones:

- n: the number of customers in the system, i.e. in the queue and being served
- w: the number of customers in the queue

- t_r: the customer response time
- t_w: the customer waiting time
- U: system utilization, that is the percentage of time the system is busy serving,
- X: throughput, i.e., the average number of customers served per unit of time.

Queueing system analysis usually evaluates the last two average performance indices and the probability distribution or the first moments of random variables n and w, and possibly t_r and t_w.

Let s denote the number of customers in service and let t_s denote the service time, where $E[t_s] = 1/\mu$ is the average service time. Then we can write $n = w+s$ and $t_r = t_w + t_s$. Let N denote the average number of customers in the system and R the mean response time, i.e., $N = E[n]$ and $R = E[t_r]$. Hence:

$$N = E[w] + E[s]$$
$$R = E[t_w] + E[t_s].$$

The operational analysis defines some interesting relations between average performance indices under general assumptions [48,49,42]. An important relation is the Little's Law that states that the average number of customers is equal to the product of the mean response time and the system throughput, i.e.:

$$N = XR \tag{6}$$
$$E[w] = XE[t_w].$$

Queueing systems are analyzed by defining an associated discrete-space continuous-time stochastic process, whose state include the system population n. Under independent and exponential assumptions we can define and associated continuous-time Markov chain whose stationary solution in terms of state probability is given by formula (4) [42]. Other performance indices can be derived by the stationary state probability and the basic relations, such as Little's law. For some queueing systems, such as the M/M/1 and M/M/m systems, the underlying Markov process is a birth-death Markov process, which yield the simple closed-form solution of the stationary state probabilities given by formula (5). Hence these queueing systems can be easily analyzed and the average performance indices show simple analytical expressions.

M/M/1. The M/M/1 queueing system has Poisson independent arrivals, exponential service time distribution, one server and FCFS discipline. Let λ denote the arrival rate, μ the service rate and let $\rho = \lambda/\mu$ denote the traffic intensity. The system state is defined by n, the customer population and the stationary state probability π_n can be computed by the associated Markov process that is a birth-death process with constant rates λ and μ. If the system satisfies the stability condition, from formula (5) we immediately obtain:

$$\pi_n = \rho^n(1-\rho) \quad n \geq 0. \tag{7}$$

The M/M/1 is stable if the $\rho < 1$, i.e., if the arrival rate is less than the service rate, $\lambda < \mu$. Hence, by the state probability and formula (7) we can derive other performance indices, such as the average population, the mean response time, system throughput and utilization as follows:

$$N = \frac{\rho}{1-\rho} \tag{8}$$

$$R = \frac{1/\mu}{1-\rho} \tag{9}$$

$$U = 1 - \pi_0 = \rho \tag{10}$$

$$X = \lambda. \tag{11}$$

M/M/m. The M/M/m queueing system extends system M/M/1 to m servers under the same exponential and independence assumption for the arrival and service processes, with arrival rate λ and service rate μ for each independent server. Let us define $\rho = \lambda/(m\mu)$. The system state is defined by the customer population as for the M/M/1 system and the associated Markov process is still a birth-death process with constant birth rate λ and variable state-dependent death rate $\mu_n = \min\{n, m\}\mu$ for state $n \geq 0$. If the system satisfies the stability condition, that is if $\rho < 1$, then by formula (5) we obtain:

$$\pi_n = \begin{cases} \frac{(m\rho)^n}{n!}\pi_0 & \text{if } 1 \leq n \leq m \\ \frac{m^m \rho^n}{m!}\pi_0 & \text{if } n > m \end{cases} \tag{12}$$

where:

$$\pi_0 = \Big[\sum_{k=0}^{m-1} \frac{(m\rho)^k}{k!} + \frac{(m\rho)^m}{m!} \frac{1}{1-\rho} \Big]^{-1}.$$

Hence we can derive other performance indices, such as the average population and the mean response time as follows:

$$N = m\rho + \pi_m \frac{\rho}{(1-\rho)^2}$$

$$R = \frac{1}{\mu} + \frac{\pi_m}{m\mu(1-\rho)^2}.$$

M/M/∞. The M/M/∞ queueing system has Poisson independent arrivals with arrival rate λ and an unlimited number of independent exponential servers, each with service rate μ. As a consequence the customers never queue and we can immediately observe that the mean response time is equal to the mean service time, i.e., $R = 1/\mu$. Let $\rho = \lambda/\mu$. The Markov process associated to M/M/∞ is a birth-death process with constant birth rate λ and variable state-dependent death rate $\mu_n = n\mu$. The system is always stable, and the stationary state probability is given by:

$$\pi_n = \frac{\rho^n}{n!} e^{-\rho} \quad k \geq 0, \tag{13}$$

from which we obtain the average population $N = \rho$.

We can apply this type of analysis to various types of M/M/1 queueing systems by defining and solving the associated Markov chain such as for example the M/M/1/B system with finite capacity B and the M/M/1//K system with finite population K. For these and similar basic queuing systems we can derive closed form expressions for the stationary state probability π_n and other average performance indices [21,42].

Queueing systems with non-exponential time distribution are in general more difficult to analyze. We now briefly recall some main results for the M/G/1 system and we refer to the literature for the detailed analysis of more complex queueing system [42,21].

M/G/1. The M/G/1 queueing system has Poisson independent arrivals with arrival rate λ, service time t_s with general distribution, one server and FCFS discipline. Let μ denote the service rate, that is $E[t_s] = 1/\mu$, and let $\rho = \lambda/\mu$. If we consider the state n, defined as the customer population, we cannot define an associated Markov chain. By assuming that the service time can be represented a probabilistic combination of exponential stages (see Section 2.3) we can still define an associated Markov process whose state includes the residual service time of the customer currently in service [42]. This Markov chain is not birth-death, and it can be analyzed by considering an embedded discrete-time Markov chain. The stationary state probability is obtained in terms of z-transform, defined as a function of ρ and of the Lapalce transform of the service time distribution. The M/G/1 is stable if the $\rho < 1$, i.e., if the arrival rate is less than the service rate, $\lambda < \mu$. The average population is obtained by the Khintchine-Pollaczek theorem [42] as follows:

$$N = \rho + \frac{\rho^2(1 + CV^2)}{2(1 - \rho)}, \tag{14}$$

where CV is the coefficient of variation of the service time distribution.

Using the basic relations and Little's law we can derive other performance indices. Note that formula (14) holds for the M/G/1 system with any queueing discipline independent on the service time and without pre-emption.
A detailed analysis of M/G/1 system can be found in [21,58].

2.3 Coxian Distribution

Many results for single queueing systems can be obtained by assuming exponential distribution for service time and arrival time. However, for modeling purposes this can be a constraint that should often be relaxed. We shall now introduce a class of quite general distributions used in the analysis of QNs. Coxian distributions are defined as a linear combination of exponential variables, and can be represented by a network of exponential stages. Coxian distribution plays an important role in QNs for two reasons: first they are used to model service time distribution for some station types (from 2 to 4) in BCMP QNs [9], as we shall see in Section 3.3. A second reason is that Coxian distribution has a rational Laplace transform and can approximate any distribution arbitrarily closely [42].

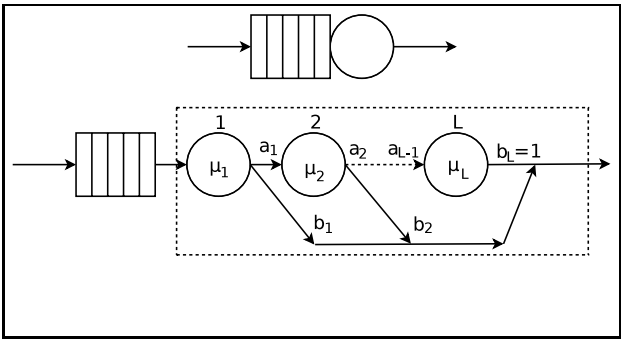


Fig. 2. A queueing system with a Coxian server with L stages

Figure 2 illustrates a queueing system whose service time is modeled by a Coxian distribution with L exponential stages. There can be at most one customer in stages 1 to L at any time. Customers enter the service via stage 1. The service time at node ℓ, $1 \leq \ell \leq L$, is exponentially distributed with mean $1/\mu_\ell$. A customer completing its service at stage ℓ leaves the system with probability b_ℓ or proceeds to stage $\ell + 1$ with probability a_ℓ, $(a_\ell + b_\ell = 1, 1 \leq i \leq L - 1)$. After stage L, the customer leaves the system with probability 1 $(b_L = 1)$. This service distribution is referred to as a Coxian distribution with L stages. Any probability distribution function can be arbitrarily closely approximated by a Coxian distribution. This framework allows an arbitrary distribution that does not have the Markovian property to be approximated by a Coxian distribution that has the Markovian property. For example, consider the service process of Figure 2 with $b_\ell = 0$, $1 \leq \ell \leq L - 1$ and $\mu_\ell = \mu$, $1 \leq \ell \leq L$. This represents the Erlang distribution with L stages. Its coefficient of variation is equal to $1/\sqrt{\lambda}$ and as $L \to \infty$, it goes to zero approximating a deterministic distribution.

A similar representation based on a network of exponential stages is defined for Phase-type distributions, that also have rational Laplace transforms and can approximate any distribution function [58].

2.4 Queueing Disciplines

Queueing system behavior and performance indices depend on various system parameters and specifically on the scheduling or queueing discipline. Customer scheduling may depend on the arrival time, like FCFS and LCFS discipline or may be independent of the time arrival and service demand, such as the Random scheduling.

Computer system processor scheduling often use Round-Robin discipline where each customer is served for a fixed quantum of time δ. If we consider the quantum size δ much smaller that the average service time then for $\delta \to 0$ we can define the Processor Sharing (PS) discipline. In a system with PS discipline and service rate μ when there are n customers waiting in the queue, each customer receives the service with rate μ/n.

The scheduling discipline where all the customers are immediately served by a free server is called Infinite Server (IS). Examples of IS or delay service centers are terminal components in timesharing computer systems. The queuing algorithm may depend on the service time required by the customer and possibly the service already given to the customer. Examples are the Shortest Processing Time First, the Shortest Remaining Processing Time First (SRPTF).

The algorithms may also depend on the customer priority that may be defined by some abstract classification of the customers or may depend on the service time. Priority discipline can be with or without pre-emption. The latter type applies priority scheduling when the server is assigned to a customer after an idle period or at the service completion, and the service is never interrupted. Pre-emption priority allows a customer with higher priority then the one currently in service to interrupt that service and to be served. Note that in this case the customers with low priority do not affect the customers behavior with high priority.

M/G/1 queueing systems with priority discipline can be analyzed to derive the mean waiting time for each customer class and for the all the classes, by considering various types of priority scheduling. By comparing priority disciplines with and without pre-emption one can observe that pre-emption improves the average mean waiting time but worsens the average service time, because of the service interruption. In general priority disciplines improve the global average waiting time with respect to non priority ones, and this improvement grows with the system congestion [48].

In the following we mainly focus on the following disciplines FCFS, LCFSPR (LCFS with pre-emptive resume, i.e., the work done for a pre-empted customer must not be repeated), PS and IS. Queueing networks whose nodes have such queueing disciplines can be efficiently analyzed under special conditions, as we describe in the next Section.

3 Queueing Networks

In this section we introduce QN models. First in Sections 3.1 and 3.2 we define the model whose analysis is based on the solution of the associated Markov stochastic process. Then in Section 3.3 we focus on the class of product-form QNs and we review BCMP theorem. Section 3.3 discusses the characterization of product-form QNs and reviews both BCMP extensions and non BCMP QNs.

3.1 Model Definition

A *queueing network* (QN) is a collection of service centers (or *stations*) that provide service to a set of customers that move among the stations. If the total number of customers, i.e., the *population*, in the network is constant then the network is *closed*. If customers may arrive from (depart to) places outside the network then the network is *open*. Examples of open and closed networks are given by Figures 8 and 9 in Section 5. Informally, a QN is defined by a set of M service centers $\Omega = \{1, \ldots, M\}$, the set of customers and the network topology.

Each service center is defined by:

- the number of servers. We usually suppose independent and identical servers;
- the service rate. Each server can serve a customer with a speed which can be either constant or dependent on the station state.
- the queueing discipline. The customers in the service center wait to be served according to the scheduling discipline, as introduced in Section 2.4.

Customers are described by:

- their total number for closed models,
- the arrival process to each service center for open models,
- the service demand to each service center. The service demand of the customer is expressed in units of service. The service rate of each server is given by units of service / units of time. Hence we usually consider the *service time* combines these two parameters as follows: let α be the customer service demand and β the server service rate, then the ratio α/β is the service time. Hereafter we consider the service time as a non-negative random variable with mean denoted by $1/\mu$.

The network topology models the customer behavior among the interconnected service centers. We assume a non-deterministic behavior represented by the following probabilistic model. In a QN with M stations, when a customer completes its service in station i it immediately exits the node and moves to station j with probability p_{ij}, with $1 \leq i, j \leq M$. For open networks the customer may also exit the QN from station i with probability p_{i0}. Then customer behavior in the QN is represented by the *routing probability matrix* $\mathbf{P} = [p_{ij}]$, $1 \leq i, j \leq M$, where $\sum_{j=1}^{M} p_{ij} = 1$ for each station i.

A QN is *well-formed* if it has a well-defined long-term customer behavior, i.e.:

- a closed QN is well-formed if every station is reachable from any other with a nonzero probability;
- for open QNs we can add a virtual station 0 that represents the external behavior, that is it generates external arrivals and absorbs all departing customers, so obtaining a closed QN. Thus an open QN is well-defined by referring to the closed QN definition.

In simple queueing models we assume that all the customers are statistically identical, i.e., the service times and the routing probabilities are independent of the customer identity. However, modelling real systems can require to identify different categories of customers that define both the service time and the routing probabilities. To this aim we introduce QNs with multiple types of customers, by defining the concepts of class and chain. In the following we use classes in the the global sense (see for example [39,9,68,48]) as opposed to the local sense (see for example [63,20]). A *chain* forms a permanent categorization of customers, i.e., a customer belongs to the same chain during its whole activity in the network. A *class* is a temporary classification of customers, i.e., a customer can switch from a class to another (usually with a probabilistic behavior) during its activity in the

network. The customer service time in each station and the routing probabilities usually depend on the class it belongs to. So we can have multiple-class single-chain QNs or multiple-class and multiple-chain QNs. In the following $\mathcal{R}$ denotes the set of classes of the QN, R the number of classes, $\mathcal{C}$ the set of chains and C the number of chains. In a well-formed QN, classes can be partitioned into chains, such that there cannot be a customer switch from classes belonging to different chains. The probabilistic behavior of customers in a well-formed QN is represented by C routing probability matrices $\mathbf{P^{(c)}}$, one for each chain c. A customer that completes its service at station i and class r, either leaves the system with probability $p^{(c)}_{ir,0}$ or immediately moves to a station j in class s with probability $p^{(c)}_{ir,js}$, $1 \leq i,j \leq M$, $r,s \in \mathcal{R}$ and r,s belongs to the same chain c. For multiple-chain QNs we distinguish *open chains*, i.e., if arrivals from and departures to external are allowed, and *closed chains*, i.e., when there is a finite number of customers. Let $K^{(c)}$ denote the population of a closed chain $c \in \mathcal{C}$ and let $p^{(d)}_{0,ir} > 0$ denote the routing probability of an external arrival to station i, class r of open chain $d \in \mathcal{C}$. A QN is said to be open if all its chains are open, closed if they are all closed, and mixed otherwise. We can apply an algorithm [39] that checks whether a multiple-class and multiple-chain QN is well-formed, given set Ω and the routing matrices, and it defines a partition of the set $E = \{(i,r) : r \in \mathcal{R}, 1 \leq i \leq M\}$ into C ergodic chains. We summarize the notation for QN classes and chains as follows:

- $\mathcal{R}$ is the set of classes and $R = |\mathcal{R}|$, $1 \leq i \leq M$,
- $\mathcal{R}_i = \{r : \exists j,s,c \ : \ p^{(c)}_{js,ir} > 0 \vee p^{(c)}_{0,ir} > 0 \ r,s \in \mathcal{R}, \ c \in \mathcal{C}, \ 1 \leq j \leq M\}$ is the set of classes served by station i, so $\mathcal{R} = \bigcup_{i=1}^{M} \mathcal{R}_i$,
- $E_c = \{(i,r) | r \in \mathcal{R}_i, 1 \leq i \leq M, \text{class } r \text{ belongs to chain } c\}$,
- $\mathcal{R}^{(c)}_i = \{r \in \mathcal{R}, (i,r) \in E_c\}$ is the set of the classes served by station i belonging to chain c, $1 \leq i \leq M$ and $1 \leq c \leq C$, so $\mathcal{R}_i = \bigcup_{c=1}^{C} \mathcal{R}^{(c)}_i$,
- $\mathcal{R}^{(c)} = \bigcup_{i=1}^{M} \mathcal{R}^{(c)}_i$ is the set of all classes belonging to chain c. We use $r^{(c)}$ to point out that a class r belongs to the set $\mathcal{R}^{(c)}$.

Example. Figure 3 shows an example of a two node multiple-class and multiple-chain QN. The QN has $R = 3$ classes and $C = 2$ chains. Chain 1 is open and formed by classes 1 and 2, while chain 2 is closed. Figure 4 sketches how to identify chains of the QN by the analysis of the strong connected components of a directed graph, whose nodes denote the couples (station, class) and arrows are determined by the QN routing matrix. Then we have $\mathcal{R} = \{1,2,3\}$, $\mathcal{R}_1 = \{1,2,3\}$, $\mathcal{R}_2 = \{1,3\}$, $E_1 = \{(1,1),(1,2),(2,1)\}$, $E_2 = \{(1,3),(2,3)\}$, $\mathcal{R}^{(1)}_1 = \{1,2\}$, $\mathcal{R}^{(1)}_2 = \{1\}$.

A chain c, $1 \leq c \leq C$, is said to be a single-class if it is has a just one class, so that there is not class switching inside the chain. In the QN of Figure 3, Chain 1 is *not* single class, while Chain 2 is single class. When all the chains of a QN are single-class, then we say that the QN is single-class and multiple-chain. In this case, the notation can be simplified, e.g., an element of the probability routing matrix $\mathbf{P}^{(c)}$, $1 \leq c \leq C$, can be written as $p^{(c)}_{ij}$ which represents the probability

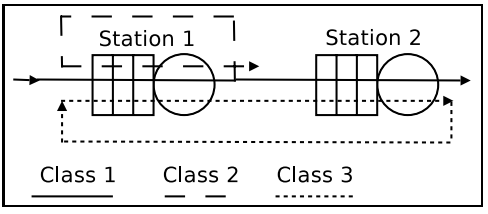


Fig. 3. Example of multiple-class and multiple-chain queueing network

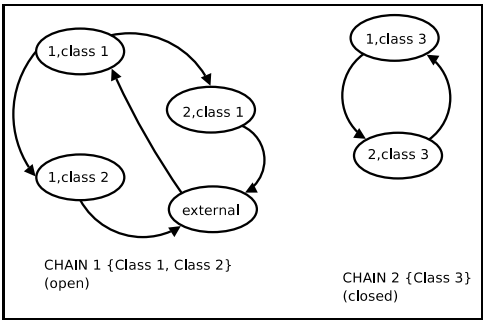


Fig. 4. Example of a class graph for a multiple-class and multiple-chain queueing network

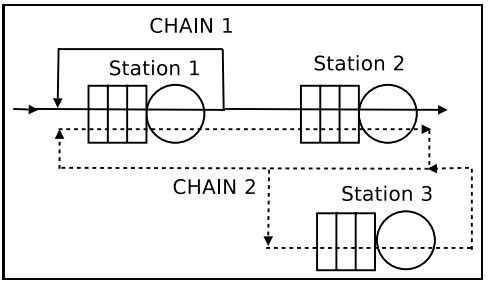


Fig. 5. Single-class and multiple-chain queueing network

for a chain c customer of going to station j after being served by station i. Figure 5 illustrates and example of single class and multiple-chain QN with two chains and three stations.

In the following we suppose that the QNs are well-formed, i.e., in the long-run no class can be empty of customers with probability 1 and no class can have an unlimited growth of the number of customers. In other word the stationary QN behavior is stable.

3.2 Markovian Queueing Networks

Consider a QN with M stations, $\Omega = \{1, \ldots, M\}$ and R classes. Let $n_{ir}(t)$ be the number of customers of class r at the station i at time t, $n_i(t) = \sum_{r=1}^{R} n_{ir}(t)$ the total number of customers at station i at time t and $\mathbf{n_i}(t) = (n_{i1}(t), \ldots, n_{iR}(t))$. Let the state of the QN at time t be $\mathbf{n}(t) = (\mathbf{n_1}(t), \ldots \mathbf{n_M}(t))$. We can associate to the QN state a discrete-space continuous-time time-homogeneous ergodic Markov chain, then we say that the QN is Markovian. As we are interested in studying the steady-state performance indices, we can ignore the time parameter t in the state definition. A Markovian QN requires independence and exponential assumptions of the random variables that represent the state. If we consider independent Poisson arrivals for open chains and exponential service time distributions whose rate can depend only on the state of the system then we can define an associated Markov chain with state $\mathbf{n}$. Let $\pi(\mathbf{n})$ denote the stationary probability of state $\mathbf{n}$ and let $\mathbf{Q} = [q_{\mathbf{n},\mathbf{n}'}]$ be the infinitesimal generator of the Markov chain, where $q_{\mathbf{n},\mathbf{n}'}$ denotes the transition rate from state $\mathbf{n}$ to state $\mathbf{n}'$. If the QN is stable, the Markov chain yields a steady-state solution. The steady state probability $\boldsymbol{\pi}$ is defined by the normalized solution of the system of global balance equations (4).

Other performance indices of the QN are derived from the stationary state distribution of the process. Unfortunately the generality of this approach is limited by its computational complexity. One can easily observe that the process state space cardinality, i.e., the number of states and of global balance equations, often makes the solution of the system intractable. More precisely, for an open network the process state space is infinite and we can obtain an exact solution only in some special cases, when the matrix $\mathbf{Q}$ shows a particular regular structure. For example some QNs with special structure can be analyzed by matrix-geometric technique [58]. For a closed network the process state space grows exponentially with the network parameters that are the number of service centers, customers and customers types. For example, for a single-class exponential QN with M service centers and K customers the state space cardinality is $\binom{M+K-1}{K}$. Hence a direct solution of the QN by the underlying Markov process becomes soon prohibitively expensive.

Note that Markovian QNs can be defined also by relaxing exponential conditions for the time distribution. By using Coxian or Phase-type distributions and by a more detailed and appropriate state definition, it is still possible to define and underlying Markov chain. However this further increases the state space complexity and its cardinality.

In the next section we introduce the class of QNs with product-form that shows a simple closed form of the stationary probability.

3.3 Product-Form Queueing Networks

Product-form QNs provide precise and detailed results in terms of performance indices such as queue length distribution, average response time, resource utilization and throughput. These performance indices are evaluated for each

component and for the overall network. Product-form network analysis is based on a set of assumptions on the system parameters that lead to a closed-form expression of the stationary state distribution. Consider a single-class and single-chain QN with M service centers. Let $\mathbf{n} = (n_1, \ldots, n_M)$ be its state, and n_i the number of customers at station i, and finally $n = \sum_{i=1}^{M} n_i$ the network population for $1 \leq i \leq M$. Product-form QNs show a closed-form of the joint queue length distribution π that is defined by the associated Markov process, as follows:

$$\pi(\mathbf{n}) = \frac{1}{G} d(n) \prod_{i=1}^{M} g_i(n_i), \tag{15}$$

where G is a normalizing constant, function d is defined in terms of network parameters and g_i is a function of n_i and depends on the type of service center i, $1 \leq i \leq M$. For open networks $G = 1$, whereas for closed networks $d(n) = 1$. Product-form QNs have been first introduced by Jackson for open QNs in [38], and by Gordon and Newell for closed QNs in [33]. Both these models require exponential service time distributions, Poisson arrival for Jackson networks, and consider only single-class and single-chain QNs. BCMP theorem [9] extends these classes of QNs to open, closed and mixed, multiple-class and multiple-chain QNs. It also considers non-exponential service time distributions for certain scheduling disciplines.

Product-form QNs can be efficiently analyzed by algorithms with a polynomial time computational complexity in the number of network components that will be presented in Section 4. This class of models allows a good balance between a relative high accuracy in the performance results and the efficiency in model analysis and evaluation. Moreover product-form networks yield several interesting properties such as insensitivity and exact aggregation that greatly influenced the application of this class of models as a powerful tool for performance evaluation. We shall now define the class of BCMP QNs, and then we discuss some properties and possible characterizations of product-form QNs.

BCMP Queueing Networks

BCMP theorem [9] characterizes a wide class of QNs with product-form. Before stating the main result of the theorem we introduce the hypothesis and the model definition. In the following we refer to a QN which satisfies the following assumptions as a BCMP QN. For the sake of clarity we first present the BCMP theorem for multiple-chain, single-class networks. In order to simplify the notation we assume that the class number and the chain number are the same, e.g., we can write $c \in R_i$ to identify a chain c served by node i. Then we consider multiple-chain and multiple-class networks.

Service center types. The network consists of M service stations:

$$\Omega = \{1, \ldots, M\}.$$

The number of classes and chains is the same $R = C$ and each chain can be open or closed. We refer to the notation introduced in Section 3.1 and summarized in Table 1. BCMP theorem considers four types of service centers:

Type 1: FCFS Service discipline and exponentially distributed and chain-independent service time.

For types 2, 3 and 4 stations, the service time distributions have rational Laplace transforms (see Section 2.3) and the average service rate can depend on the state of each customer chain.

Type 2: PS Service discipline.
Type 3: IS service centers.
Type 4: LCFSPR Service discipline.

We first assume a constant service rate. Let $\mu_i^{(c)}$ denote the service rate of station i for chain c customers, $1 \leq i \leq M$ and $c \in \mathcal{C}$. For type 1 service centers $\mu_i^{(c)} = \mu_i$, as the service time is chain independent.

State vector. BCMP theorem gives a product-form solution for states with different levels of detail. Let $\mathbf{n} = (\mathbf{n_1}, \ldots, \mathbf{n_M})$ denote the network state, where:

- $\mathbf{n_i} = (n_i^{(1)}, \ldots, n_i^{(C)})$ is the occupancy vector at station i, $1 \leq i \leq M$,
- $n_i = \sum_{c=1}^{C} n_i^{(c)}$ is the number or customers at station i, $1 \leq i \leq M$,
- $n = \sum_{i=1}^{M} n_i$ is the total number of customers in the network,
- $n^{(c)} = \sum_{i=1}^{M} n_i^{(c)}$ is the number of customers of chain c in the network, $1 \leq c \leq C$. Note that $n^{(c)} = K^{(c)}$ if c is a closed chain.

Arrivals to open chains. For open chains, customers arrive to the network from an external source. There are two possible state dependencies for the arrival process. In the first case the total arrival process to the network is a Poisson process with parameter $\lambda(n)$ where n is the total number of customers in the network. Arrivals are distributed among the classes according to the routing probabilities. Let $p_{0i}^{(c)}$ denote the probability of an external arrival to node i and open chain c, then by decomposition property of Poisson processes, the arrival process to station i and chain c is a Poisson process with rate $\lambda(n)p_{0i}^{(c)}$, with $\sum_{i=1}^{M} \sum_{c \in \mathcal{R}_i} p_{0i}^{(c)} = 1$. In the second case the arrival processes to different open chains are independent Poisson processes whose rates depend on the total number of customers of the associated chain, i.e., $\lambda_c(n^{(c)})$ where $1 \leq c \leq C$ and c is an open chain. The arrival process to station j, $1 \leq j \leq M$ is a Poisson process with rate $\lambda_c(n^{(c)})p_{0j}^{(c)}$. Routing probabilities satisfies the normalizing constant $\sum_{i=1}^{M} p_{0i}^{(c)} = 1$. For a closed chain c we set $p_{0i}^{(c)} = 0$ for every station $i = 1, \ldots, M$ and let $K^{(c)}$ denote the constant chain population, i.e., $n^{(c)} = K^{(c)}$ for all states $\mathbf{n}$.

Traffic equations. We first define the set of expected number of visits or (relative) throughput for each node i and chain c, denoted by $e_i^{(c)}$. These values are obtained as the solution of the following C systems of linear equations, called traffic equations:

$$e_j^{(c)} = \sum_{i=1}^{M} e_i^{(c)} p_{i,j}^{(c)} + p_{0,j}^{(c)} \quad j = 1, \ldots, M \;\; 1 \leq c \leq C \tag{16}$$

These systems uniquely define the solution $e_j^{(c)}$ if chain c is open, while they are not uniquely determined if chain c is closed. If chain c is open $e_j^{(c)}$ represents the expected number of visits (visit ratio) for a customer of chain c at station i. If chain c is closed, then we replace one equation for a station i such that $\mathcal{R}_i^{(c)} \neq \emptyset$ with $e_i^{(c)} = 1$, and we obtain a set of linear independent equations. Then the solution $e_j^{(c)}$ represents the relative visit ratio of a customer belonging to chain c to station j for each visit to station i. Let us define the $\rho_i^{(c)} = e_i^{(c)}/\mu_i^{(c)}$ for each station $i = 1, \ldots, M$ and chain $c \in \mathcal{R}_i^{(c)}$, let $\rho_i^{(c)} = 0$ if $c \notin \mathcal{R}_i^{(c)}$.

We now state the BCMP theorem:

Theorem 1 (BCMP theorem, single-class, multiple-chain [9]). *Let Ω be a BCMP QN under stability conditions. Then the following steady state probability holds:*

$$\pi(\mathbf{n}) = \frac{1}{G} d(\mathbf{n}) \prod_{i=1}^{M} g_i(\mathbf{n_i}), \tag{17}$$

where $d(\mathbf{n}) = \prod_{a=0}^{n-1} \lambda(a)$ if the arrival rate depends on the total number of customer in the system, or $d(\mathbf{n}) = \prod_{c=0}^{C} \prod_{a=0}^{n^{(c)}-1} \lambda_c(a)$. Functions $g_i(\mathbf{n_i})$ are determined as follows:

- *For type 1, type 2 and type 4 stations:*

$$g_i(\mathbf{n_i}) = n_i! \Big[\prod_{c=1}^{c} \frac{1}{n_i^{(c)}!} (\rho_i^{(c)})^{n_i^{(c)}} \Big], \tag{18}$$

  considering $\mu_i^{(c)} = \mu_i$ for type 1 stations.
- *For type 3 stations:*

$$g_i(\mathbf{n_i}) = \prod_{c=1}^{C} \frac{1}{n_i^{(c)}!} (\rho_i^{(c)})^{n_i^{(c)}}. \tag{19}$$

Proof. The proof given in [9] is based on a detailed definition of the network state and by substitution of the product-form expression into the global balance equations of the Markov continuous-time process underlying the QN. If i is a type 1 station then its state is represented by vector $\mathbf{b_i} = (b_{i1}, \ldots, b_{in_i})$ where n_i is the number of customers in the station and b_{ij} is the number of the class of the j-th customer in the FCFS order. If i is a type 2 or 3 station $\mathbf{b_i} = (\mathbf{b_{i1}}, \ldots, \mathbf{b_{iR}})$

where $\mathbf{b_{ir}}$ is a vector whose components represent the number of customers of class r at a certain stage of service. If i is a type 4 station, its state is similar to the one introduced for type 1 stations but each vector component always stores the customer service stage (note that the discipline has resume feature). In order to prove the theorem it is shown that:

1. The stationary probability distribution for the detailed state (that we omit for the sake of clarity) is correct by substitution on the global balance equations system.
2. The stationary probability distribution for the state $\mathbf{n}$ defined above can be obtained as marginal distribution of the aggregation of the detailed states.

□

Let us now consider stations with load-dependent service rates. BCMP theorem identifies three kinds of state-dependent service rates:

Type A: The service rate of a customer at station i depends on the total number of customers n_i. Let $x_i(n_i)$ be a positive function which gives the relative service rate at station i, i.e., $x_i(1) = 1$, such that the actual service rate for a class r customer at station i is $x_i(n_i)\mu_{ir}$. Function x_i is also called capacity function. Then function g_i defined by equation (18) or (19) must be multiplied by factor:

$$\prod_{a=1}^{n_i} \frac{1}{x_i(a)}.$$

Type B: The service rate of a class r customer at station i depends on $n_i^{(c)}$. Define $y_i^{(c)}(n_i^{(c)})$ as a positive capacity function, similarly to x_i definition in the previous case. Then function g_i definition must be multiplied by factor:

$$\prod_{c \in \mathcal{R}_i} \prod_{a=1}^{n_i^{(c)}} \frac{1}{y_i^{(c)}(a)}.$$

Note that this state-dependent service rate violates the BCMP hypothesis for type 1 stations, that is it applies only for types 2, 3 and 4.

Type C: The service rate of a customer at station i depends on the number of customers in several stations. Let $\mathcal{H} \subseteq \Omega$ be a subset of the station set, and $n_H = \sum_{h \in \mathcal{H}} n_h$. Define $z_H(n_H)$ to be a positive capacity function which represents the relative service rate when $n_H = 1$. Then the product $\prod_{h \in \mathcal{H}} g_i(\mathbf{n_i})$ in equation (17) becomes:

$$\Big[\prod_{h \in \mathcal{H}} g_i(\mathbf{n_i}) \Big] \prod_{a=1}^{n_H} \frac{1}{z_H(a)}.$$

Note that state-dependent service rates can be combined giving a great flexibility to BCMP networks expressive power. For example in order to model a PS (or LCFSPR) with different mean service rates for class, with m constant rate servers, it suffices to set $x_i(n_i) = \min\{m, n_i\}/n_i$ and $y_i^{(c)}(n_i^{(c)}) = n_i^{(c)}$.

BCMP theorem for multiple-class and multiple-chain QNs. We now state a more general form of the BCMP theorem, as presented in [9], for QNs where a customer can move within a chain by class switching. In order to represent class switching we have to modify the notation as follows:

- The routing matrices $\mathbf{P}^{(c)}$ assume the general form $p^{(c)}_{ir,js}$ for nodes i, j, classes r and s in chain c as introduced in Section 3.1.
- Service rate at station $i = 1, \ldots, M$ is relative to the class (for types 2, 3 and 4 stations), so we use $\mu^{(c)}_{ir}$,
- There are C independent traffic equation systems which define the (relative) visit ratio for class and station, $e^{(c)}_{ir}$ for $i = 1, \ldots, M$ and $r \in \mathcal{R}^{(c)}_i$, $1 \leq c \leq C$, as follows:

$$e^{(c)}_{jr} = \sum_{i=1}^{M} \sum_{s \in \mathcal{R}^{(c)}_i} e^{(c)}_{is} p^{(c)}_{is,jr} + p^{(c)}_{0,jr}. \tag{20}$$

  Let us define $\rho^{(c)}_{ir} = e^{(c)}_{ir}/\mu^{(c)}_{ir}$ for station i, class r belonging to chain c.
  Example. Consider the multiple-class network of Figure 3. Assuming a constant arrival rate λ, the traffic equations system for the open chain 1 is:

$$\begin{cases} e^{(1)}_{11} = 1 \\ e^{(1)}_{12} = e^{(1)}_{11} p^{(1)}_{11,12} \\ e^{(1)}_{21} = e^{(1)}_{11} p^{(1)}_{11,21} \end{cases},$$

  which has a unique solution. The traffic equation system for Chain 2 has infinite linear dependent solutions, and we can set $e^{(2)}_{21} = 1$ that yields $e^{(2)}_{22} = 1$. For an open chain $e^{(c)}_{ir}$ represents the mean number of visits at station i with class r, for a closed chain it represents the average number of visits at station i with class r relative to a visit for an arbitrary couple (station, class) belonging to the same chain of c.
- State $\mathbf{n} = (\mathbf{n_1}, \ldots, \mathbf{n_M})$ components are the occupancy vectors for classes, i.e., $\mathbf{n_i} = (\mathbf{n_i^{(1)}}, \ldots, \mathbf{n_i^{(C)}})$ where $\mathbf{n_i^{(c)}}$ is a vector whose components $n^{(c)}_{ir}$ represent the number of customers of class r (belonging to chain c) at station i, for $r \in \mathcal{R}^{(c)}_i$ and $1 \leq i \leq M$ and $1 \leq c \leq C$.

Then for a multiple-chain and multiple-class QN, Theorem 1 still holds by using the following definitions of function g_i:

$$g_i(\mathbf{n_i}) = n_i! \prod_{c=1}^{C} \prod_{r \in \mathcal{R}^{(c)}_i} \frac{1}{n^{(c)}_{ir}!} (\rho^{(c)}_{ir})^{n^{(c)}_{ir}}, \tag{21}$$

for a service center i of type 1, 2 and 4, and:

$$g_i(\mathbf{n_i}) = \prod_{c=1}^{C} \prod_{r \in \mathcal{R}^{(c)}_i} \frac{1}{n^{(c)}_{ir}!} (\rho^{(c)}_{ir})^{n^{(c)}_{ir}}, \tag{22}$$

for a type 3 service center i.

These QNs can have load-dependent service centers. Capacity function for state-dependent service rate of type B can be reformulated considering the class population instead of the chain population at the node, i.e., the capacity function can be $y_{ir}^{(c)}$ for station i, and $r \in \mathcal{R}_i^{(c)}$.

In order to evaluate the QN performance indices we can simplify the computation of the BCMP product-form solution as follows:

Aggregation by chain. If Ω is a multiple-class and multiple-chain QN, under certain assumptions we can compute the steady state probability on an aggregate state. Consider an aggregated vector $\mathbf{n_a}$ where $n_{ai}^{(c)} = \sum_{r \in \mathcal{R}_i} n_{ir}^{(c)}$ for $i = 1, \ldots, M$ and $c = 1, \ldots, C$. In the general case, as the service rate can depend on the class of the customer, we can express the aggregated probability $\pi_a(\mathbf{n_a})$ as sum of probabilities $\pi(\mathbf{n})$ as follows:

$$\pi_a(\mathbf{n_a}) = \sum_{\substack{\mathbf{n} | \sum_{r \in \mathcal{R}_i^{(c)}} n_{ir}^{(c)} = n_{ai}^{(c)} \\ i=1,\ldots,M \wedge c=1,\ldots,C}} \pi(\mathbf{n}).$$

If for each station the service rate is load-independent or depends on the network state according to type A, C or B (formulated on the chain population and not the class population), then BCMP theorem for multiple-class and multiple-chain QNs can be simplified to single-class and multiple-chain case. In fact formulas (17), (18) and (19) hold by considering the (relative) visit ratio at station i, $1 \leq i \leq M$ and chain c as the sum of the visit ratios for all the classes belonging to the same chain as station i: $e_i^{(c)} = \sum_{r \in \mathcal{R}_i^{(c)}} e_{ir}^{(c)}$, and the service rate per chain is the weighted sum of the service rate of the classes belonging to the same chain: $\mu_i^{(c)} = \sum_{r \in \mathcal{R}_i^{(c)}} e_{ir}^{(c)} \mu_{ir}^{(c)} / e_i^{(c)}$.

Aggregation for open networks by node population. If all chains of the network are open, arrival rates λ are constant and capacity functions $x_i(n_i)$ are of type A, then it is possible to simplify the steady state distribution function for the aggregated state $\mathbf{n}' = (n_1, \ldots, n_M)$ where n_i is the total number of customer at station i, $n_i = \sum_{c=1}^{C} \sum_{r \in \mathcal{R}_i^{(c)}} n_{ir}^{(c)}$. In fact it is possible to study each station in isolation considering $\rho_i = \sum_{c=1}^{C} \sum_{r \in \mathcal{R}_i^{(c)}} \lambda \cdot \rho_{ir}^{(c)}$ and capacity function x_i. This results provide a convenient way for normalizing probabilities and checking if the stability condition holds (i.e. $\forall i = 1, \ldots, M : \rho_i < 1$).

Characterization of Product-Form Queueing Networks

Product-form QN allows to obtain a set of performance indices without generating and solving the associated Markov process and the system of global balance equations. The characterization of the classes of product-form QNs is an interesting task. Under some assumptions (e.g., non-priority scheduling, infinite queue capacity, non-blocking factors, state-independent routing probabilities) it

is possible to give conditions on the service time distributions and on the station queueing disciplines to determine whether a well-formed QN yield a BCMP-like product-form solution. We consider the following properties which are strictly related to product-form: *local balance*, $M \implies M$, *quasi-reversibility*, *station balance*.

Local balance property. This property states that the effective rate at which the system leaves state ξ due to a service completion of a chain r customer at station i, equals the effective rate at which the system enters state ξ due to an arrival of chain r customer to station i. This result can be also generalized for multiple-class and multiple-chain networks. It can be shown that, for some queueing disciplines, local balance holds even when service times distributions are represented by a network of exponential stages [59,17]. In this case the state must include the stage at which a customer is being served. Note that:

- If a steady state probability distribution $\boldsymbol{\pi}$ satisfies the local balance equations (LBEs), then it satisfies also the global balance equations, but opposite is not true, i.e., LBEs are a sufficient condition for network solution.
- Solving LBEs is computationally easier than solving global balance ones even if it still requires to handle the set of reachable states (which can be a problem for open chains or networks). However if we need to prove that a steady state formula is correct, it can be simpler to check if it verifies LBEs.
- The local balance is a property of a station embedded in a QN. The states we are considering are still states of the network.

***M* $\implies$ *M* property.** This property is introduced in [55] and it is defined for a single queueing system. An open queueing system holds this property if under independent Poisson arrivals per class of customers, the departure processes are also independent Poisson processes. Consider an isolated station with R classes and a state space Γ. Customers of class r arrive to the system according to independent Poisson processes with rate λ_r. Let $\pi(\xi)$ be the steady state probability of state ξ with $\xi \in \Gamma$, let $|\xi|_r$ be the number of customers of class r in the station when the state is ξ. Define the set $\mathcal{S}_r^+ = \{\xi' : |\xi'|_r = |\xi|_r + 1\}$. Then the $M \implies M$ property holds if:

$$\forall \xi \in \Gamma \quad \sum_{\xi' \in \mathcal{S}_r^+} \frac{\pi(\xi') q_{\xi'\xi}}{\pi(\xi)} = \lambda_r, \tag{23}$$

where $q_{\xi'\xi}$ is the transition rate between state ξ' and ξ. Note that:

- $M \implies M$ property considers the station in isolation. Thus it can be used to decide whether a station with specific queueing discipline and service time distribution, can be embedded in a product-form QN (see for example [1,50]). If each station of a QN has the $M \implies M$ property then it has a product-form solution.

– If a network (a chain) is open and each of the station has the $M \implies M$ properties, than the network itself has the $M \implies M$ property [55].
– In a QN with service centers with non-priority scheduling disciplines property $M \implies M$ holds for every station if and only if local balance holds [39,10].

Quasi-reversibility property. A queueing system exhibits quasi-reversibility property if the queue length at a given time t is independent of the arrival times of customers after t and of departure times of customer prior to t. It is possible to prove (e.g., in [40]) that a QN whose stations are quasi-reversible yields a product-form solution. Note that:

– Quasi-reversibility property is defined for isolated stations.
– It is easy to prove (see for example [39]) that all arrival streams to a quasi-reversible system should be independent and Poisson, and all departure streams should be independent and Poisson. In other words a system is quasi-reversible if and only if it exhibits the $M \implies M$ property.

Station balance property. This property is introduced in [17] and discussed in [59] and [18]. A scheduling discipline holds station balance property if the service rates at which the jobs in a position of the queue are served are proportional to the probability that a job enters this position. We call these kinds of scheduling disciplines *symmetric disciplines*. Note that also the property is defined for an isolated station, and it is a sufficient condition for product-form (e.g., FCFS does not yields station balance).

Figure 6 shows the relation between these properties:

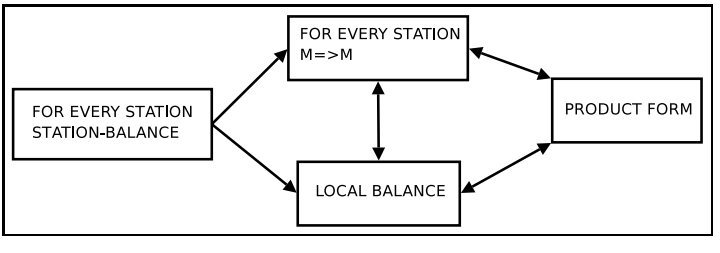


Fig. 6. Relations between properties related to product-form for nonpriority service centers

It is worthwhile trying to characterize product-form QNs with higher level properties, e.g., on properties of scheduling discipline. We can summarize some observations:

– Every service center with: A) a queueing discipline which is work-conserving and independent of the service time requirement of the customers, and B) an exponentially distributed service time, yields local balance property.

- For symmetric disciplines the QN steady state probabilities only depend on the mean of the service time distribution and on the value of the (relative) visit ratio. This is know as the *insensitivity property* of product-form networks [17]. Moreover we can say that the routing matrix influences the QN performance parameters only for the computation of the visit ratios, i.e., two networks with the same stations and different routing matrices are equivalent if they have the same (relative) visit ratio for each station.
- The symmetric disciplines are the only ones that give product-form solution if the service time distribution is not exponential [18].
- A symmetric discipline starts serving a customer as soon as it enters in the system, i.e., symmetric discipline are always preemptive discipline.

Other Product-Form Queueing Networks

Various extensions of the class of BCMP product-form networks have been proposed. They include state-dependent routing [44,75,11], i.e., the definition of routing probabilities are special functions that may depend on the state of the entire network or of subnetworks and/or single service centers. This allows representing systems with more complex features such as dynamic load balancing algorithms or adaptive routing strategies. Such models usually assume some additional constraints on the network parameters and a special structure of the routing state-dependent functions. For example Towsley [75] considered closed QNs where the routing for some service centers may be a rational function of the queue length of the service centers belonging to a downstream subnetwork with a particular topology, called parallel subnetwork. Boucherie and VanDijk have proposed an extension to more complex state-dependent routing by considering a more detailed definition of routing functions dependent on the state of subnetworks called clusters and the state of service centers [11]. The model assumes that the service centers are partitioned into a set of subnetworks that are linked by a state-dependent routing. Then the routing function between two service centers i and j that respectively belong to two disjoint subnetworks I and J has the following expression: $p_{i0}(I)p'_{IJ}p_{0j}(J)$, where $p_{i0}(I)$ and $p_{0j}(J)$ are routing functions internal to subnetworks I and J, respectively, and p'_{IJ} denotes the routing between subnetworks. This model can be useful to represent hierarchical and decomposable systems. Extensions of BCMP networks to different service disciplines have been derived. Le Boudec proved product-form solution for QNs with multiple-server nodes with concurrent class of customers that allow to represent special systems [50].

Various special classes of non BCMP QNs have been proved to have product-form solution under particular constraints. These QNs may represent some special system characteristics, such as for example, finite capacity queues, population constraints and positive and negative customers.

QNs with finite capacity queues, subnetwork population constraints and blocking have product-form solution in some special cases [2,5,34,78]. Various blocking types that describe different behaviors of customer arrivals at full capacity service centers and the servers' activity in the network have been defined.

For several special combinations of network topology, types of service centers and blocking mechanisms one can derive a product-form solution for the stationary state distribution. Moreover, one can derive various equivalence properties between product-form networks with and without blocking and between networks with different blocking type, as discussed in [5].

Another extension of QNs with product-form is the class of networks proposed by Gelenbe [29] (G-networks) with positive and negative customers that can be used to represent special system behaviors [30]. For example negative customers may represent commands to delete some transactions in databases or in a distributed computer system due to inconsistency or data locking. A negative customer arriving to a service center reduces the total queue length by one if the queue length is positive and it has no effect otherwise. Negative customers do not receive service. A customer moving between service centers can become either negative or remain positive. Such a QN has product-form solution under exponential and independence assumptions and with a Markovian routing and the solution is based on a set of non linear traffic equations of the customers. G-networks also deal with multiple-class of customers [27]. Some further extensions have been introduced in order to extend the model power of G-networks, such as the introduction of reset-customers [31], or network state-dependent service rate and routing intensities, triggered batch signal movement [28].

Product-form solution has been extended to QNs with batch arrivals and batch services [35,36] that are also related to discrete time QN models. The model evolution is described by a discrete-time Markov chain and assumes special expressions for the probability of batch arrivals and departures and correlated batch routing. The product-form solution is based on a generalized expression of the traffic equations and the quasi-reversibility property of the network. The product-form solution holds for continuous-time and discrete time QNs.

3.4 Flow-Equivalent Aggregation

An insensitivity property of product-form QNs is the exact flow-equivalent aggregation. QNs aggregation, or Norton's theorem, allows substituting a subnetwork with a simple service center so that the aggregated network is equivalent to the original one, in terms of performance indices. QNs aggregation is based on aggregation of the underlying Markov chain. Aggregation of states on Markov chains has been widely studied in order to find the steady state probability of the process by solving a reduced number of global balance equations. In [41] the authors analyze the concept and the condition of *lumpability* for finite states Markov chains. Aggregation for decomposable Markov chains is studied in [25].

For product-form QNs it is possible to apply aggregation at a higher abstraction level, i.e., subnetwork aggregation instead of state aggregation. This technique can be used to study complex QN models and in hierarchical modeling. With *exact aggregation* of product-form QNs we can replace a subnetwork with a single station with appropriately chosen service rate function so that the performance measures of remaining nodes are the same. For non-product-form QNs in general aggregation defines an approximate model of the original one

[16]. Various approximation techniques are based on aggregating subnetworks as we discuss in the next section. Consider a single-chain QN. Figure 7-A shows a QN Ω partitioned in two subnetworks. We want to replace the subnetwork Ω_a in the network Ω with an single station a. We apply exact aggregation to obtain the new network Ω^* where subnetwork Ω_a is replaced by a new station a, as shown in Figure 7-C. Exact aggregation or Norton's theorem defines the parameters of station a, called *aggregated station*, i.e., the station which replaces the subnetwork. We usually define station a as a FCFS exponential station (BCMP type 1) or, by insensitivity property, we can assume PS discipline. Let $\mu_a(n)$ denote the load-dependent service rate where there are n customers at station a. The aggregated network parameters, i.e., $\mu_a(n)$ and the routing probabilities of network Ω^*, are derived by the solution of subnetwork Ω_a analyzed in isolation, as shown in Figure 7-B. This isolated network is obtained from the original one by shorting out, that is, by setting to zero all the service times of the stations in $\Omega - \Omega_a$. The isolated network is analyzed by product-form algorithms to calculate the network throughput denoted by $\Lambda_a(n)$ when there are n customer in the network. Hence we state Norton's theorem:

Theorem 2 (Norton's theorem). *Let network Ω^* be defined as network Ω by substituting subnetwork Ω_a with a single service center a, and with same parameters for subnetwork $\Omega - \Omega_a$ and with the same number of customers K. Let node a service rate be defined as $\mu_a(n) = \Lambda_a(n)$ for $1 \leq n \leq K$. Then Ω and Ω^* are equivalent in terms of stationary state distribution of the non aggregated subnetwork.*

Exact aggregation allows to compute performance measures for subnetwork $\Omega - \Omega_a$ and for node a representing the aggregated subnetwork Ω_a. It holds for any subnetwork, i.e., for subnetworks with multiple entry and exit points and for which we have also to define a new routing matrix for the aggregated network [6], for multiple-chain networks [68,43] and for mixed networks and occupancy vector-dependent capacities [43].

Exact aggregation can be used in hierarchical system analysis. In a bottom-up system design process we can relate the performance indices of the network models at different levels in the hierarchy [49]. Exact aggregation provides a tool to define an equivalent aggregated model at the higher level of abstraction. Similarly, in a hierarchical top-down system design with predefined performance requirements, we can apply the inverse process called disaggregation or synthesis of the network to define a more detailed model with the same performance indices [7]. The disaggregation process answer the question of what the system topology and parameter should be in order to achieve the given performance goal.

An important application of exact aggregation for product-form QNs is the definition of various approximate methods for non product-form networks [51] that we discuss in Section 4.4. These algorithms are usually based on an iterative scheme and they basically apply the aggregation theorem to non product-form networks, although in this case it provides only approximate results. At each iteration step several subnetworks are analyzed and aggregated in the flow-equivalent service centers. This principle has been applied for the approximate

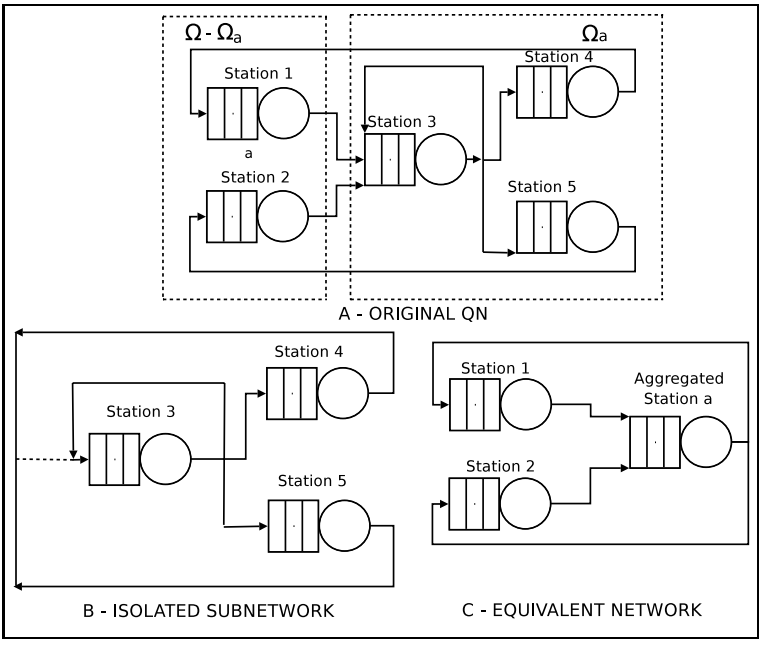


Fig. 7. Example of exact aggregation

analysis of various types of non product-form networks, such as for example networks with simultaneous resource possession and finite capacity queues.

4 Solution Algorithms for Queueing Networks

The main advantage of product-form QNs is that several efficient algorithms have been developed for their performance analysis. As a consequence efficient and powerful performance evaluation tools based on product-form QNs have been developed and applied to obtain performance indices for large networks with many service centers and customers [68,48,49,66,14].

In order to evaluate performance indices we can use either exact algorithms or approximate methods. The choice depends on a number of factors, e.g., the number of classes and chains of the network. We shall now review the most used algorithms for BCMP product-form QNs. The two main solution algorithms are Convolution and MVA. Convolution (or Buzen's Algorithm) was first introduced for single-chain networks by Buzen in [13] and then extended to multiple-chain networks by Bruell and Balbo in [12]. Mean Value Analysis (MVA) algorithm has been developed by Reiser an Lavenberg [64,62] both for single and multiple-chain networks. Another solution algorithm is the Local Balance based computation of Normalization Constant (LBANC) developed by Sauer and Chandy in [67]. A detailed discussion on various computational algorithms can be found in [68]. For multiple-chain QN several further algorithms have been proposed: Recursion

by chain (RECAL) [23], Tree-Convolution [46], Tree-MVA [77,37], Distributed Analysis by Chain (DAC) [26] and Mean Value Analysis by Chain (MVAC) [22].

We first illustrate Convolution algorithm both for single-chain and multiple-chain QNs. In Section 4.2 we present MVA for single-chain and multiple-chain QNs, then Section 4.3 briefly reviews the other algorithms paying special attention to RECAL.

4.1 The Convolution Algorithm

In this section we first describe Convolution algorithm for a single-class closed QN and then for the single-class multiple-chain closed QN. Consider a BCMP QN with the set of stations $\Omega = \{1, \ldots, M\}$, K customers and where the station service rates can depend only on the number of customers at that station (BCMP type A capacity function). The computation of the stationary state distribution $\boldsymbol{\pi}$ requires the evaluation of the normalizing constant G in formula (17). Convolution algorithm is based on a direct and efficient computation of G. Without loss of generality we assume that the first D stations have constant service rate IS discipline (simple delay stations), then the stations from $D+1$ to $D+I$ have load-independent service rates (simple stations) and the stations from $D+I+1$ to $D+I+L=M$ have the load-dependent service rates. Let $G_i(k)$ denote the normalizing constant for the QN considering a population of k customers and the first i stations. Then the network normalizing constant in formula (17) is given by $G = G_M(K)$ and is defined as follows:

$$G = G_M(K) = \sum_{\mathbf{n}|\sum_{i=1}^{M} n_i = K} \prod_{i=1}^{M} g_i(n_i), \tag{24}$$

where functions $g_i(n_i)$ is defined in Theorem 1. Direct computation of G by formula (24) takes an exponential time in M and K, i.e., it is proportional to the state space cardinality. Convolution algorithm avoids this direct computation and evaluates G recursively and it is based on the Convolution theorem [13]. In order to calculate $G_M(K)$ efficiently we write the following recursive equation:

$$G_i(k) = \sum_{n=0}^{k} G_{i-1}(k-n) g_i(n) \quad 1 \le i \le M,\ 1 \le k \le K, \tag{25}$$

with initial conditions: $G_0(k) = 0, \forall k > 0$ and $G_i(0) = 1$ for $1 \le i \le M$. Each recursive step for station i in formula (25) requires $\mathcal{O}(K^2)$ operations. This scheme can be further simplified as follows. As the first D service centers are simple delay stations then we can immediately write:

$$G_D(k) = \frac{1}{k!}\Big[\sum_{i=1}^{D} \rho_i\Big]^k, \quad 0 \le k \le K, \tag{26}$$

where ρ_i is defined in Theorem 1, Section 3.3. This formula requires $\mathcal{O}(K+D)$ operations. If i is a simple station then we can write $g_i(k) = \rho_i^k$. Hence formula (25) reduces to:

$$G_i(k) = G_{i-1}(k) + \rho_i G_i(k-1) \quad 0 \le k \le K. \tag{27}$$

Thus adding a load-independent station requires $\mathcal{O}(K)$ operations. Algorithm 1 summarizes the computation of the normalizing constant G with the Convolution algorithm by formula (26) for simple delay stations, (27) for load-independent stations and (25) for load-dependent ones.

Several performance measures can be directly evaluated by function G. For a simple station j we can write the marginal queue length distribution as follows:

$$Pr(n_j = n|K) = \rho_j^n \frac{G_M(K-n) - \rho_j G_M(K-n-1)}{G_M(K)} \; 0 \le n \le K, \tag{28}$$

that is the probability of n customers in station j, given a network population of K customers. Then we obtain the average performance indices, i.e., the mean queue length, the throughput and the utilization for node j as follows:

$$N_j = \sum_{k=1}^{K} \rho_j^k \frac{G_M(K-k)}{G_M(K)}, \tag{29}$$

$$X_j = e_j \frac{G_M(K-1)}{G_M(K)}, \tag{30}$$

$$U_j = \frac{X_j}{\mu_j}. \tag{31}$$

For a service center j with load-dependent service rate we cannot apply formula (28) and we have the following expression for the queue length distribution:

$$Pr(n_j = n|K) = g_j(n) \frac{G_{\Omega-\{j\}}(K-n)}{G_M(K)}, \tag{32}$$

where $G_{\Omega-\{j\}}$ denotes the normalizing constant of the network obtained by the original network Ω with station j removed. Hence, Convolution algorithm efficiency is reduced when the network has several load-dependent service centers.

A limitation of this algorithm is its potential numerical instability, i.e., the possible overflow or underflow in the computation of constant G. Some scaling techniques to overcome this problem have been proposed [45].

Algorithm 1 is Convolution algorithm to compute the normalizing constant, and Algorithm 2 illustrates the computation of a set of performance indices. Note that, for closed networks with L load-dependent service centers we need to apply Algorithm 1 L times, but we can use some optimization techniques that keep function G_{D+I} for simple and delay stations. The overall computational complexity of the algorithm is $\mathcal{O}(K + IK + L^2K^2)$. If all the stations have load-independent service rate the time complexity is simply $\mathcal{O}(MK)$.

Let us consider a multiple-chain QN. Let $\mathbf{K} = (K^{(1)}, \ldots, K^{(C)})$ be the population vector per chain where C is the number of closed chains, and $K^{(c)}$ is the population of chain c for $1 \le c \le C$. Equation (25) becomes:

$$G_i(\mathbf{K}) = \sum_{k_i^{(1)}=0}^{K^{(1)}} \cdots \sum_{k_i^{(C)}=0}^{K^{(C)}} G_{i-1}(\mathbf{K} - \mathbf{k_i}) g_i(\mathbf{k_i}),$$

Algorithm 1. Convolution algorithm: calculate normalizing constant G

Assumptions: The set of nodes Ω is ordered as follows: nodes from 1 to D are simple delay stations, from $D+1$ to $D+I$ are load-independent stations, from $D+I+1$ to $D+I+L=M$ are load-dependent.

{Initialization}
$G_i(0) = 1, \quad i = 1 \ldots, M$
$G_0(k) = 0 \quad k = 0, \ldots, K$
Compute visit ratios $\mathbf{e}$ by equation (116) and scale to reduce numeric problems
{Add simple delay stations}
$\rho_D = 0$
for $i = 1$ to D **do**
  $\rho_D = \rho_D + \rho_i$
end for
for $k = 1$ to K **do**
  $G_D(k) = 1/k!(\rho_D)^k$
end for
{Add simple stations}
for $m = D+1$ to $D+I$ **do**
  for $k = 1$ to K **do**
    $G_m(k) = G_{m-1}(k) + \rho_m G_m(k-1)$
  end for
end for
{Add load-dependent stations}
for $m = D+I+1$ to M **do**
  for $k = 1$ to K **do**
    $G_m(k) = 0$
    for $n = 0$ to k **do**
      $G_m(k) += G_{m-1}(k-n)g_i(n)$
    end for
  end for
end for
return G {Return matrix G}

which can be simplified for simple stations as follows:

$$G_i(\mathbf{K}) = G_{i-1}(\mathbf{K}) + \sum_{c=1}^{C} \rho_i^{(c)} G_i(\mathbf{K} - \mathbf{1_c}),$$

where $\mathbf{1_c}$ is the unit vector, i.e., all components are 0 but the c-th which is 1 and $\rho_i^{(c)}$ is defined in Section 3.3. As the first D stations are simple delay stations we can directly calculate $G_D(\mathbf{K})$:

$$G_D(\mathbf{K}) = \Big[\sum_{i=1}^{D} \rho_i^{(1)}\Big]^{K^{(1)}} \cdots \Big[\sum_{i=1}^{D} \rho_i^{(C)}\Big]^{K^{(C)}} \frac{1}{K^{(1)} \cdots K^{(C)}}.$$

Algorithm 2. Convolution Algorithm: calculate performance indices from G

Assumptions: The set of nodes Ω is ordered as follows: nodes from 1 to D are simple delay station, from $D+1$ to $D+I$ are load-independent stations, from $D+I+1$ to $D+I+L=M$ are load-dependent.
{Calculate performance indices for simple delay stations}
for $i=1$ to D **do**
calculate throughput X_i
end for
{Calculate performance indices for simple stations}
for $i=D+1$ to $D+I$ **do**
calculate N_i by formula (29)
calculate X_i by formula (30)
calculate U_i by formula (31)
end for
{Calculate performance indices for load-dependent stations} $L=0$
for $i=D+I+1$ to M **do**
$L=L+1$
if $L>1$ **then**
Let Ω' be Ω with station i as last station
Calculate G' for Ω'
Calculate performance indices using formula (30) and (32) using G'
else
Calculate performance indices using formula (30) and (32) using G
end if
end for

If $\mathbf{n_i}$ is the occupancy vector at station i, for a load-dependent station i, the the analogue of equation (32) is the following:

$$Pr(\mathbf{n_i}=\mathbf{n}|\mathbf{K})=\frac{G_{\Omega-\{i\}}(\mathbf{K}-\mathbf{n})}{G_M(\mathbf{K})}g_i(\mathbf{n}),$$

that is the probability of state $\mathbf{n}$ for station i, given the network population vector $\mathbf{K}$. The average performance indices for each station i and each chain c can be calculated as follows:

$$X_i^{(c)}=e_i^{(c)}\frac{G_M(\mathbf{K}-\mathbf{1_c})}{G_M(\mathbf{K})}$$
$$U_i^{(c)}=\rho_i^{(c)}\frac{G_M(\mathbf{K}-\mathbf{1_c})}{G_M(\mathbf{K})}$$
$$N_i^{(c)}=\sum_{a=1}^{K^{(c)}}\sum_{\mathbf{n}:n^{(c)}=a}Pr(\mathbf{n_i}=\mathbf{n})$$

Moreover the computation of G can be further simplified as presented in [54]. Let $\mathbf{k_i}=(k_i^{(1)},\ldots,k_i^{(C)})$ be a vector representing the number of customers for each chain c in the first i stations of the network, i.e., $k_i^{(c)}$ is the overall number

of customers of chain c at nodes $1,\ldots,i$: $k_i^{(c)} = \sum_{j=1}^{i} n_j^{(c)}$. Then $\mathbf{k_M} = \mathbf{K}$. The following recursive formula holds:

$$G_i(\mathbf{k_i}) = \sum_{\forall \mathbf{k_{i-1}}, \mathbf{k_{i-1}} + \mathbf{n_i} = \mathbf{k_i}} G_{i-1}(\mathbf{k_{i-1}}) g_i(\mathbf{n_i}),$$

where $\mathbf{n_i}$ represents a valid occupancy vector at station i. Noting that $G_M(\mathbf{K}) = G_M(\mathbf{k_M})$ we have the normalizing constant.

The time computational complexity for multiple-chain Convolution algorithm depends on the *product* of chain populations. Let us define:

$$H = \prod_{c=1}^{C} (K^{(c)} + 1). \tag{33}$$

Then for a QN with C chains and population vector $\mathbf{K}$, an iteration step of Convolution for a simple station requires $\mathcal{O}(CH)$ operations and for a load-dependent station requires $\mathcal{O}(H^2)$ operations. For the special case of a QN where all the chains have the same population $K^{(c)} = \mathcal{K} = K/C$, and all the stations are load-independent, then the time complexity is $\mathcal{O}(MC\mathcal{K}^C)$. Thus the time requirements grows rapidly with the number of chain and their population. The Convolution algorithm suffers of numerical instability with problem of underflow, overflow and round-off errors.

4.2 Mean Value Analysis (MVA)

Algorithm MVA directly calculates the QN performance indices avoiding the explicitly computation of the normalizing constant. It is based on the *arrival theorem* [47,70] and on Little's theorem. We briefly recall the arrival theorem:

Theorem 3 (Arrival theorem). *In a closed product-form QN the steady state distribution of the number of customers at station i at customer arrival times at i is identical to the steady state distribution of the number of customers at the same station at an arbitrary time with that user removed from the QN.*

Consider a single-chain QN where we number the stations such that those from 1 to D are simple delay stations, from $D+1$ to $D+I$ are load-independent service rate stations (simple stations), and from $D+I+1$ to $D+I+L=M$ are load-dependent stations. By Theorem 3 MVA derives the following recursive scheme:

For simple delay stations:

$$\begin{aligned}
R_j(K) &= \tfrac{1}{\mu_j} && 1 \le j \le D \\
R_j(K) &= \tfrac{1}{\mu_j}(1 + N_j(K-1)) && D+1 \le j \le D+I \\
X_j(K) &= \frac{K}{\sum_{i=1}^{M} (e_i/e_j) R_i(K)} && 1 \le j \le D+I \\
N_j(K) &= X_j(K) R_j(K) && 1 \le j \le D+I.
\end{aligned} \tag{34}$$

For load-dependent service rate stations:

$$\begin{array}{ll} R_j(K) = \sum_{n=1}^{K} \frac{n}{\mu_j(n)} Pr(n_j = n-1|K-1) & D+I+1 \le j \le M, \\ & K > 0 \\ X_j(K) = \frac{K}{\sum_{i=1}^{M}(e_i/e_j)R_i(K)} & D+I+1 \le j \le M \\ Pr(n_j = n|K) = \frac{X_j(K)}{\mu_j(n)} Pr(n_j = n-1|K-1) & D+I+1 \le j \le M, \\ & 1 \le n \le K, K > 1 \\ Pr(n_j = 0|K) = 1 - \sum_{n=1}^{K} Pr(n_j = n|K) & \end{array} \tag{35}$$

with the initial conditions $Pr(n_j = 0|0) = 1$ and $N_j(0) = 0$.

Note that MVA does not compute the normalizing constant G so avoiding the consequent numerical instability. However, the computation of marginal probability $Pr(n_j = 0|K)$ can lead to numerical problems as it tends to zero. To overcome this drawback of potential numerical instability, a modified algorithm (MMVA) has been proposed. MMVA algorithm modifies the recursive scheme of formula (35) for the zero probability as follows:

$$Pr(n_j = 0|K) = Pr(n_j = 0|K-1)\frac{X_i(K)}{X_i^{\Omega-\{j\}}(K)}, \tag{36}$$

where $X_i^{\Omega-\{j\}}$ is the throughput of any node i computed for the QN with node j removed. This direct computation improves the algorithm numerical stability. On the other hand MMVA has a higher computational complexity than MVA. In fact a QN with L load-dependent stations requires to solve $2^L - 1$ additional QNs. For a single-chain QN without load-dependent stations with K customers and M nodes, MVA has the same time complexity of Convolution, i.e., $\mathcal{O}(KM)$. If the QN has only load-dependent stations, then MVA complexity becomes $\mathcal{O}(MK^2)$ which is better than Convolution complexity $\mathcal{O}(M^2K^2)$. However, consider that to overcome the numerical instability problem MMVA has the same complexity as Convolution.

Consider now a multiple-chain QN and let $\mathbf{K} = (K^{(1)}, \ldots, K^{(C)})$ be the population of the network by chain. The multiple-chain MVA algorithm defines the following recursive scheme:

– for simple delay stations and simple stations, for $1 \le c \le C$ and $1 \le j \le D + I$:

$$\begin{array}{ll} R_j^{(c)}(\mathbf{K}) = \frac{1}{\mu_j^{(c)}} & 1 \le j \le D \\ R_j^{(c)}(\mathbf{K}) = \frac{1}{\mu_j^{(c)}}[1 + N_j^{(c)}(\mathbf{K} - \mathbf{1_c})] & K^{(c)} > 0, \\ & D+1 \le j \le D+I \\ X_j^{(c)}(\mathbf{K}) = \frac{K^{(c)}}{\sum_{i=1}^{M}(e_i^{(c)}/e_j^{(c)}) \cdot R_i^{(c)}(\mathbf{K})} & \\ N_j^{(c)}(\mathbf{K}) = X_j^{(c)}(\mathbf{K}) R_j^{(c)}(\mathbf{K}) & \end{array} \tag{37}$$

– for load-dependent stations (of type A BCMP form), let $K = \sum_{d=1}^{C} K^{(d)}$, $1 \leq c \leq C$ and $D + I + 1 \leq j \leq M$:

$$\begin{aligned} R_j^{(c)}(\mathbf{K}) &= \sum_{n=1}^{K} \frac{n}{\mu_j^{(c)}(n)} Pr(n_j = n-1|\mathbf{K} - \mathbf{1_c}) \qquad K^{(c)} > 0 \\ X_j^{(c)}(\mathbf{K}) &= \frac{K^{(c)}}{\sum_{i=1}^{M} (e_i^{(c)}/e_j^{(c)}) R_i^{(c)}(\mathbf{K})} \\ Pr(n_j = n|\mathbf{K}) &= \sum_{c:K^{(c)}>0} \frac{X_j^{(c)}(\mathbf{K})}{\mu_j^{(c)}(n)} Pr(n_j = n-1|\mathbf{K}) \; n > 0 \\ Pr(n_j = 0|\mathbf{K}) &= 1 - \sum_{n=1}^{K} Pr(n_j = n|\mathbf{K}) \end{aligned} \tag{38}$$

with the initial condition $N_j^{(c)}(\mathbf{0}) = 0$ and $Pr(n_j = 0|\mathbf{0}) = 1$ for each station j and chain c.

Algorithm 3 sketches the MVA algorithm for multiple-chain single-class QNs assuming that all quantities are zero at negative populations.

Multiple-chain MVA algorithm has of the same numerical instability problem as single-chain MVA and it can be solved in a similar mode. Let H be defined by equation (33). Then the time complexity of the algorithm is $\mathcal{O}(KCH)$ operations for an iteration step on a load-dependent station and $\mathcal{O}(CH)$ for simple stations. MVA complexity for load-dependent stations is lower than Convolution algorithm, that is $\mathcal{O}(H^2)$. Indeed Convolution does not take advantage of chain independent capacity functions, i.e., basic MVA considers only capacity functions of type A, while Convolution considers both type A and B. For a network without load-dependent stations, MVA and Convolution have the same time complexity, i.e., $\mathcal{O}(MC\mathcal{K}^C)$ assuming that all the chains have the same population $\mathcal{K} = K/C$. In [73] MVA algorithm is generalized in order to calculate higher moments of performance measures.

4.3 Other Algorithms for Multiple-Class Queueing Networks

Recursion by Chain Algorithm (RECAL) has a similar approach to Convolution one since it computes the normalizing constant in order to obtain the mean performance measures of the product-form QN. However, as the name says, the recursion is based on the chains of the QN. RECAL algorithm is very well suited for networks with a large number of job classes but a small number of stations [22,24]. The recursive scheme is based on the formulation of the normalizing constant G for C chains as function of the normalizing constant for $C-1$ chains.

Consider a QN with $C > 1$ chains and M load-independent service rate stations. The algorithm is based on a QN transformation into a new dual QN by augmenting the number of chains so that each chain has just one customer. Therefore it partitions each chain c into $K^{(c)}$ identical subchains with one job per chain. The recursive algorithm applies to compute the normalizing constant of the dual network. The average performance indices of the original QN are obtained by those of the dual one. In the following we use the up-script $*$ to denote the fictitious network parameters, e.g., Ω^* is the fictitious network itself, $C^* = K^* = K$ is the number of chains and customers. Let us number the customers and the corresponding chain in the dual network with labels $1, \ldots, K$,

Algorithm 3. MVA algorithm for multiple-chain QNs

Assumptions: The set of nodes Ω is ordered as follows: nodes from 1 to D are simple delay stations, from $D+1$ to $D+I$ are load-independent stations and from $D+I+1$ to $D+I+L=M$ are load-dependent.

{Initialization}
for $i=1$ to M **do**
  for $c=1$ to C **do**
    $N_i^{(c)}(\mathbf{0})=0$
  end for
  if $i>M_s$ **then**
    $P_i(0|\mathbf{0})=1$
  end if
end for
{Obtain performance indices}
{The following cycle iterates H times!}
for $\mathbf{k}=\mathbf{0}$ to $\mathbf{K}$ **do**
  for $c=1$ to C **do**
    for $i=1$ to D **do**
      $R_i^{(c)}=1/\mu_i^{(c)}$
    end for
    for $i=D+1$ to $D+I$ **do**
      $R_i^{(c)}(\mathbf{k})=\frac{1}{\mu_i^{(c)}}[1+N_i^{(c)}(\mathbf{k}-\mathbf{1_c})]$
    end for
    for $i=D+I+1$ to M **do**
      $R_i^{(c)}(\mathbf{k})=\sum_{n=1}^{K}\frac{n}{\mu_i^{(c)}(n)}P(n-1|\mathbf{k}-\mathbf{1_c})$
    end for
    {Let $j|e_j^{(c)}>0$, calculate the cycle time to j}
    $CT_j^{(c)}=0$
    for $i=1$ to M **do**
      $CT_j^{(c)}=CT_j^{(c)}+e_i^{(c)}/e_j^{(c)}R_i^{(c)}(\mathbf{k})$
    end for
    for $i=1$ to M **do**
      $X_i^{(c)}=(e_i^{(c)}/e_j^{(c)})(K^{(c)}/CT_j^{(c)})$
      $N_i^{(c)}=R_i^{(c)}(\mathbf{k})\cdot X_i^{(c)}(\mathbf{k})$
    end for
  end for
  for $i=D+I+1$ to M **do**
    for $n=1$ to k **do**
      $P_i(n|\mathbf{k})=0$
      for $c=1$ to C **do**
        $P_i(n|\mathbf{k})=P_i(n|\mathbf{k})+U_i^{(c)}/x_i(n)\cdot P_i(n-1|\mathbf{k}-\mathbf{1_c})$
      end for
    end for
    $P_i(0|\mathbf{k})=1-\sum_{n=1}^{k}P_i(n|\mathbf{k})$
  end for
end for

hence customer k is the only one that belongs to the k-th chain in the dual network. Let function $\nu(k)$ denote the chain index of k-th customer in the original QN.

Let $G^*_c(\mathbf{v_c})$ denote the normalizing constant of the dual network for the first c chains, $1 \leq c \leq C^*$, where $\mathbf{v_c} = (v_{1c}, \ldots, v_{Mc})$ is to be defined. We have that $G_{C^*}(\mathbf{0})$ is the normalizing constant for the dual QN and it corresponds to the normalizing constant of the original network, i.e., $G^*_{C^*}(\mathbf{0}) = G$. RECAL applies the following recursive scheme:

$$G^*_d(\mathbf{v}_d) = \sum_{i=1}^{M}(1 + v_{id}\delta_i)\frac{e_i^{(\nu(d))}}{\mu_i^{(\nu(d))}}G^*_{d-1}(\mathbf{v}_d + \mathbf{1_i}), \tag{39}$$

for $d = 1 \ldots, K^*$ and $\mathbf{v}_d \in \mathcal{F}_d$, with:

$$\mathcal{F}_d = \{\mathbf{v}_d | v_{id} \geq 0 \ \text{ for } i = 1, \ldots M, \sum_{i=1}^{M} v_{id} = K^* - d\} \tag{40}$$

$$\delta_i = \begin{cases} 1 & \text{if } i \text{ is of type 1, 2, 4} \\ 0 & \text{if } i \text{ is of type 3} \end{cases} \tag{41}$$

The initial condition is $G^*_0(\mathbf{v}_0) = 1$ for all $\mathbf{v}_0 \in \mathcal{F}^*_0$. RECAL then computes the performance indices for the last chain C^* of the dual network stations, denoted by $X_i^{*(C^*)}$ and $N_i^{*(C^*)}$, as follows:

$$X_i^{*(C^*)} = \begin{cases} e_i^{(\nu(C^*))} \sum_{j=1}^{M} \frac{G^*_{C^*-1}(\mathbf{1_j})}{G^*_{C^*}(\mathbf{0})(M+K^*-1)} & \text{if there are no IS nodes} \\ e_i^{(\nu(C^*))} \frac{G^*_{C^*-1}(\mathbf{1_j})}{G^*_{C^*}(\mathbf{0})} & \text{if } j \text{ is any of the IS nodes} \end{cases} \tag{42}$$

$$N_i^{*(C^*)} = \frac{G^*_{C^*-1}(\mathbf{1_i})}{G^*_{C^*}(\mathbf{0})} \cdot \frac{e_i^{(\nu(C^*))}}{\mu_i^{(\nu(C^*))}}.. \tag{43}$$

Then the original network performance indices for node i and class $c = \nu(K^*)$ are obtained as follows:

$$X_i^{(c)} = K^{(c)} X_i^{*(C^*)} \tag{44}$$

$$N_i^{(c)} = K^{(c)} N_i^{*(C^*)} \tag{45}$$

In order to obtain the performance indices for any other chain $c' = 1 \ldots, C$ of the original network, we have to relabel the customers and modify function ν such that $\nu(C^*) = c'$. Algorithm 4 sketches the RECAL method. Note that the computation is optimized by an appropriate ordering of the dual network chains, by choosing the last C customers belonging to different chains of the original QN. RECAL has been extended to QNs with load-dependent station whose capacity function is of type A [24]. Note that if $K^{(c)} = \mathcal{K}$ for all the chains c, M and $\mathcal{K}$ constant, one can prove that for $C \to \infty$ the time requirement is $\mathcal{O}(C^{M+1})$ [23].

RECAL algorithm computes performance measures through the recursive computation of the normalizing constant. This can give the same numerical

problems as discussed for Convolution algorithm. MVAC (Mean Value Analysis by Chain) [22] and DAC (Distribution Analysis by Chain) [26] implements, like RECAL, a recursion scheme based on chains, but with a direct computation of some performance parameters. Consequently these algorithms are numerically robust, even for load-dependent stations.

If the network is sparse, i.e., most of the chains visit just a small number of the QN stations, one should consider the use of a tree algorithm. Tree-MVA [77,37] and Tree-Convolution [46] are extensions of algorithm MVA and Convolution. They are designed to give their best performance for sparse networks. The main idea is to build a tree data structure where QN stations are leaves, that are combined into subnetworks in order to obtain the full QN (depicted by the root of the tree). Tree algorithms used for dense QNs can give worse performance than the corresponding linear algorithms. Tree-Convolution has the same numerical properties as standard Convolution, while Tree-MVA is more robust than standard MVA.

The algorithms presented in previous sections do not allow class switching. For networks with class switching, it is possible to reduce a closed QN with C ergodic chains and class switching to an equivalent closed network with C chains without class switching, as proved in [53].

The algorithms defined for closed QNs can be used also for mixed QNs. One approach consists in modifying the station capacity functions in order to account the service capacity consumed by the open chains. Then one analyzes the remaining closed QN. Thus, the complexity of solving a mixed QN basically depends on the number and the population of closed chains (see [39,48] for details).

4.4 Approximate Algorithms

Large product-form QNs can be also analyzed by approximate algorithms in order to reduce the computational complexity. In fact time (but also space) complexity of exact algorithms increases quickly for large networks with many customers. This is especially true for multiple-chain models. Two approximate algorithms for product-form QN are based on MVA. The Bard and Schweitzer Approximation, introduced in [8,69] is a popular and widely applied approximate algorithm [60]. Another algorithm is Self-Correcting Approximation Technique (SCAT), introduced in [57], extended in [3] and generalized as the Linearizer Algorithm in [19].

The basic idea of these algorithms is to approximate the MVA recursive scheme and apply an approximate iterative scheme. Consider a multiple-chain single-class QN, and let $N_i^{(c)}(\mathbf{K})$ be the mean population at station i, chain c when the total network population is $\mathbf{K} = (K^{(1)}, \ldots, K^{(C)})$ for $1 \leq i \leq M$ and $1 \leq c \leq C$. MVA recursive equations (37) requires to compute $N_i^{(c)}(\mathbf{K} - \mathbf{1_d})$ for all $d = 1 \ldots, C$ to obtain $N_i^{(c)}(\mathbf{K})$. The Bard-Schweitzer Approximation algorithm approximates the average queue length as follows:

$$N_i^{(c)}(\mathbf{K} - \mathbf{1_d}) = \frac{|\mathbf{K} - \mathbf{1_d}|_c}{K^{(c)}} N_i^{(c)}(\mathbf{K}), \tag{46}$$

Algorithm 4. RECAL algorithm for multiple-chain QNs

Assumptions: Ω is a QN with M load-independent stations.

1: Number the jobs of the dual network Ω^*, and define an appropriate function ν such that the last C jobs belong to different chains in the original QN.
2: $G_0^*(\mathbf{v_0}) = 1$ for all $\mathbf{v_0} \in \mathcal{F}_0$ defined by (40)
{Consider the first $K^* - C$ nodes}
3: **for** $c = 1$ to $K^* - C$ **do**
4: **for** $\mathbf{v_c} \in \mathcal{F}_c$ **do**
5: Compute $G_c^*(\mathbf{v_c})$ by formula (39)
6: **end for**
7: **end for**
{Compute performance indices}
8: **for** $d = 1$ to C **do**
9: Number the jobs of the dual network Ω^*, and modify the last C values of ν such that $\nu(K^*) = d$
10: **for** $c = K^* - C + 1$ to K^* **do**
11: **for** $\mathbf{v_c} \in \mathcal{F}_c$ **do**
12: Compute $G_c^*(\mathbf{v_c})$ by formula (39)
13: **end for**
14: **end for**
15: Compute the performance indices by equations (44) and (45)
16: **end for**

where

$$|\mathbf{K} - \mathbf{1_d}|_c = \begin{cases} K^{(c)} & \text{if } c \neq d \\ K^{(c)} - 1 & \text{if } c = d \end{cases}.$$

Then by substituting this expression in formula (37), MVA becomes an iterative approximate algorithm. The iteration stops if the value of $N_i^{(c)}(\mathbf{K})$ in two consecutive iteration steps differ by less than a chosen error ϵ. This approximation technique assumes that removing a customer from chain c only affects the mean number of customer $N_i^{(c)}$. There are examples of systems where this assumption is not acceptable.

SCAT algorithm iterative scheme is similar to that of Bard-Schweitzer Approximation, but it estimates the change in the mean number of customers for two successive iteration steps. This estimate is used to modify equation (46). This technique for closed QNs with load-independent stations gives better results than Bard-Schweitzer Approximation thanks to its self-correcting capability, but it provides less precise results for QNs with load-dependent service rate stations. The *Extended SCAT Algorithm* presented in [3] improves the approximation accuracy. Discussion on approximate MVA based algorithms comparison, accuracy, uniqueness and convergence are presented in [10,39,60].

Some other approximate algorithms are based on the so-called *summation method* that defines an approximate function f_i which relates the throughput of a stations with the mean number of customers, i.e., $N_i = f_i(X_i)$. This method can be used both for single-chain and multiple-chain QNs [10].

5 Examples

In this section we illustrate some examples of application of QN modeling for system performance evaluation. The purpose of this section is just to give the reader an idea on how it is possible to obtain analytical performance measures from a QN based stochastic model. The whole process of performance analysis and prediction of real systems is a complex task which involves measurements and models validations [52,39] and is out of the scope of this work.

Example 1 (Machine repair model). This classical example considers a system composed of α identical machines which can achieve the same task with identical speed. They operate independently, in parallel and are subject to breakdown. At most $\beta \leq \alpha$ of them can be operating simultaneously (active). An active machine operates until failure. The active-time is a random time exponentially distributed with mean $\frac{1}{\mu_1}$. After a failure a machine waits for being repaired. At most γ machines can be in repair, and the repair-time is a random time exponentially distributed with mean $\frac{1}{\mu_2}$. Figure 8 illustrates the QN that represents the system.

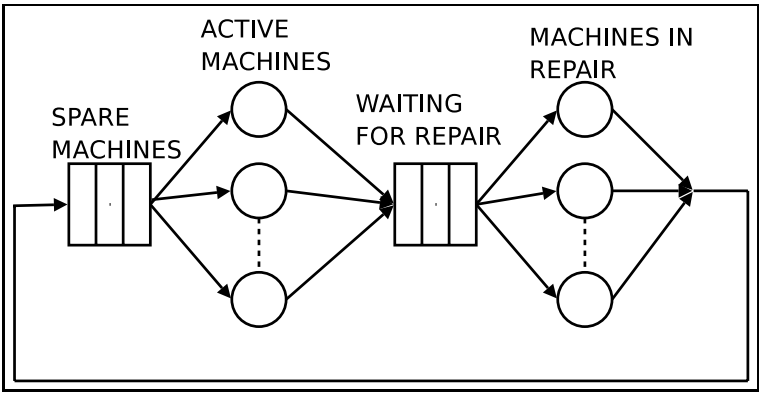


Fig. 8. Repairman model

We can study the model as a single-chain and single-class closed BCMP QN consisting of $M = 2$ stations, where station 1 represents the state of operative machines, and station 2 the machines in repair. The routing matrix is $\mathbf{P} = [[0, 1], [1, 0]]$ for which we immediately obtain visit ratios e_1 and e_2. Station server rates are μ_1 for station 1 servers and μ_2 for station 2 servers. Both the system have multiple servers, and there are β servers for node 1 and γ for node 2. We use the BCMP representation with a single server with load-dependent service rate whose capacity function is:

$$x_1(n_1) = \begin{cases} n_1 & \text{if } n_1 \leq \beta \\ \beta & \text{if } n_1 > \beta \end{cases}$$
$$x_2(n_1) = \begin{cases} n_2 & \text{if } n_2 \leq \gamma \\ \gamma & \text{if } n_2 > \gamma \end{cases}$$

QN population is $K = \alpha$. Then we can apply Convolution or MVA algorithm for single-chain QNs described in Section 4 to compute the following performance measures:

- steady state probability distribution, that is the probability of n_1 active machines and n_2 machines in waiting to be repaired,
- mean number of working machines, and mean number of machines in repair, corresponding to measures N_1 and N_2,
- the utilization at station 1, that is the ratio between the effective average work and the maximum work, i.e., when there are always β active machines. Note that for efficiency purposes one should desire a $U_1 \to 1$ and $U_2 \to 0$.
- mean time that a machine is broken, i.e., the response time R_2 of node 2.

Note that by the exponential assumptions, we can choose any BCMP-type queueing disciplines to compute the product-form solution in a single-chain and single-class QN. In fact in this case, disciplines PS, LCFSPR and FCFS are equivalent in terms of steady state probability computation. We can model more complex active and repair time by assuming appropriate Coxian service time distribution that can be used to approximate a wide class of random distributions. Then we obtain a BCMP QN if we assume PS or LCFSPR queueing discipline.

Example 2 (Database mirror). Consider a database (DB) repository system with two servers. The first server is the master, and the second one is a mirror. At the query arrival a dispatcher routes the query to the primary or to the secondary database. When a query is sent to the mirror, if the required data are found then the answer is sent back to the client. If the required data are not found, then the mirror redirects the query to the primary database that provides the answer. The system is analyzed to identify the optimal dispatcher routing strategy that gives the lowest system response time, given the DB average service times, the cache hit probability for the slave database, and the arrival rate.

Under some independence and exponential assumptions we can model the system by an open BCMP QN where a request is a customer. The QN is formed by $M = 3$ stations and is illustrated in Figure 9. We assume a probabilistic behavior of the requests that arrives according to a Poisson process with rate λ. The first station corresponds to the dispatcher which routes the requests either to the master (with probability p) or to the mirror (with probability $1 - p$). We assume that the routing is request independent so to define a simple chain QN, otherwise we should use a multiple-chain QN. A request can be fulfilled by the mirror with probability q and it generates a new request to the master with probability $1 - q$. We assume that the dispatcher is a delay station, i.e., the requests never queue and they are dispatched almost instantaneously. In order

to obtain a BCMP QN if we model DB stations with PS queueing discipline we can relax the hypothesis on exponential service time distribution. If we assume a FCFS discipline, database stations must have an exponentially distributed service time.

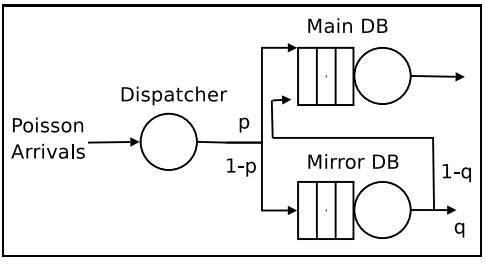


Fig. 9. Open QN modeling the system of Example 2

This open BCMP QN can be easily studied. We immediately derive the visit ratios e_i, $i = 1, 2, 3$ by solving the traffic equations be formula (16) that yield $e_1 = 1$, $e_2 = p + (1-p)(1-q)$, $e_3 = 1-p$. Then by setting $\rho_i = \lambda e_i/\mu_i$ where μ_i is the mean service rate of station i, for $i = 1, 2, 3$, we apply the BCMP formulas given in Section 3 to derive the performance indices. For example the average response time for each node i given by formula (9) and the overall QN average response time is $R = R_1 + e_2 R_2 + e_3 R_3$. Then we can apply a parametric analysis of response time R as function of probability p in order to identify the optimal routing strategy.

Example 3 (Network performance from the client view). Consider a typical system architecture used by a company to provide an Internet connection to a number of machines. The intranet is connected to a 10Mbps LAN, with an access to Internet through a Proxy server and then to a router/firewall. Suppose that the company has doubled the number of clients in the last year, and it registered an impressive slow-down of the network performace (i.e., the response time for an Internet request grew up to three times). The goal of the performance analysis is to find a possible bottleneck in the system and suggest a solution.

Figure 10 illustrates a QN model to study the performance indices from the client-side of an Internet access. In this example [52] a set of clients access the Internet through a LAN and a Proxy server. If the Proxy server has the request cached then it answers to the client through the LAN connection, otherwise the request goes to router. Note that, in order to distinguish client answers from client requests, since they both pass through the LAN connection, we use class switching among the same chain. Therefore the model is a multiple-class and single-chain QN. The QN is formed by $M = 7$ service centers, $R = 2$ classes, $C = 1$ chains. We order the classes of the QN for all nodes, first for class 1 and then for class 2, and we denote by p be the proxy cache hit probability, and $q = 1 - p$, then the QN routing matrix is:

$$\mathbf{P} = \begin{array}{c|c|c} & & \\ (1,1) & 0\,1\,0\,0\,0\,0\,0 & 0\,0\,0\,0\,0\,0\,0 \\ (2,1) & 0\,0\,1\,0\,0\,0\,0 & 0\,0\,0\,0\,0\,0\,0 \\ (3,1) & 0\,0\,0\,q\,0\,0\,0 & 0\,p\,0\,0\,0\,0\,0 \\ (4,1) & 0\,0\,0\,0\,1\,0\,0 & 0\,0\,0\,0\,0\,0\,0 \\ (5,1) & 0\,0\,0\,0\,0\,1\,0 & 0\,0\,0\,0\,0\,0\,0 \\ (6,1) & 0\,0\,0\,0\,0\,0\,0 & 0\,0\,0\,0\,0\,0\,1 \\ (7,1) & 0\,0\,0\,0\,0\,0\,0 & 0\,0\,0\,0\,0\,0\,0 \\ \hline (1,2) & 0\,0\,0\,0\,0\,0\,0 & 0\,0\,0\,0\,0\,0\,0 \\ (2,2) & 1\,0\,0\,0\,0\,0\,0 & 0\,0\,0\,0\,0\,0\,0 \\ (3,2) & 0\,0\,0\,0\,0\,0\,0 & 0\,0\,0\,0\,0\,0\,0 \\ (4,2) & 0\,0\,0\,0\,0\,0\,0 & 0\,1\,0\,0\,0\,0\,0 \\ (5,2) & 0\,0\,0\,0\,0\,0\,0 & 0\,0\,0\,0\,0\,0\,0 \\ (6,2) & 0\,0\,0\,0\,0\,0\,0 & 0\,0\,0\,0\,0\,0\,0 \\ (7,2) & 0\,0\,0\,0\,0\,0\,0 & 0\,0\,0\,1\,0\,0\,0 \end{array}$$

System performance analysis based on the QN allows the analyst to derive:

- The effect of the Proxy server on the overall system performance. By relating the cache hits and the response time with the cache size, one could decide the Proxy server characteristic for a given amount of requests.
- The behavior of the client response time, given the system parameters, as function of the number of requests.
- The effect of the outgoing and incoming link on the performance of the network. This is usually determined by the company agreements with the ISP.

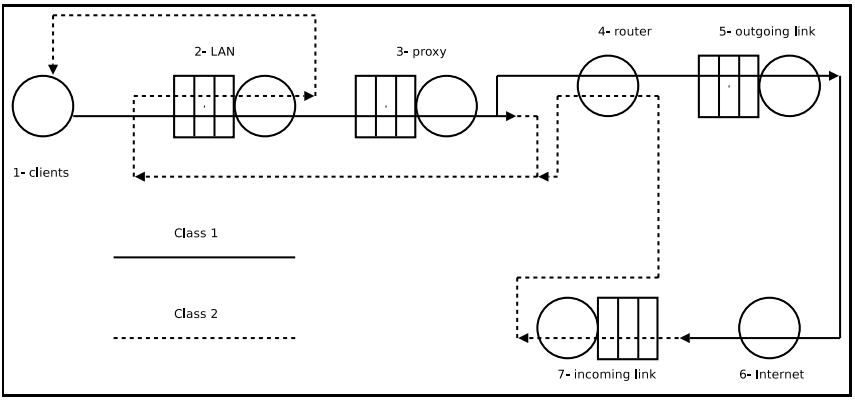


Fig. 10. QN associated to the system described in Example 3

The complete definition of the QN model requires the specification of the service center characteristics and traffic load. We assume that a workload characterization method has been used to define the workload specification. Then we have to define the service time distribution and its mean for each node and class

Table 1. Glossary for queueing network symbols

Symbol	Meaning
C	Number of chain of the QN
$\mathcal{C}$	Set of the QN chains
E	Set of couple (i, r) for each station i and each class $r \in \mathcal{R}_i$
E_c	Set of couple (i, r) for each station i and each class $r \in \mathcal{R}_i$ belonging to chain c
$e_i^{(c)}$	(Relative) Visit ratio to node i, chain c
$e_{ir}^{(c)}$	(Relative) Visit ratio to node i, class r belonging to chain c
G	Normalizing constant for closed product-form QN
$g_i(n_i)$	Function associated to node i in BCMP theorem
$\mathbf{K}$	Vector $(K^{(1)} \dots K^{(C)}$ in a closed QN
$K^{(c)}$	Number of customer in closed chain c
K	Total number of customers in a closed QN
$\mathbf{n}$	An (aggregated) state of a QN
$\mathbf{n_i}$	Occupancy vector at node i for chain
$n_i^{(c)}$	Number of users belonging to chain c at node i of a QN
$\mathbf{n_i^{(c)}}$	Occupancy vector at node i for classes of chain c
$n_{ir}^{(c)}$	Number of customers of class r belonging to chain c at station i
n_i	Total number of users at station i
n	Number of customers in the QN
M	Number of stations in a QN
N_i	Mean number of users at node i
$N_i^{(c)}$	Mean number of users at node i belonging to chain c
$\mathbf{P}$	Routing probabilities matrix of the single-chain QN
$\mathbf{P}^{(c)}$	Routing probabilities matrix of chain c of the QN
$p_{ij}^{(c)}$	Probability of going to node j from node i for a customer of chain c in single-class multiple-chain QNs
$p_{ir,js}^{(c)}$	Probability of going to node j and class s from node i form class r for a customer of chain c
R	Number of classes of the QN
$\mathcal{R}$	Set of classes of the QN
R_i	Mean response time at node i
$R_i^{(c)}$	Mean response time at node i for chain c customers
$\mathcal{R}$	Set of classes of the QN
$\mathcal{R}_i$	Set of classes served by service center i
$\mathcal{R}_i^{(c)}$	Set of classes served by service center i belonging to chain c
U_i	Utilization of node i
$U_i^{(c)}$	Utilization of node i, chain c
X_i	Throughput of node i
$X_i^{(c)}$	Throughput of node i for chain c customers
λ	Total arrival rate to the QN
$\lambda^{(c)}$	Total arrival rate to open chain c of the QN
$\rho_i^{(c)}$	ratio $e_i^{(c)}/\mu_i^{(c)}$
$\rho_{ir}^{(c)}$	ratio $e_{ir}^{(c)}/\mu_{ir}^{(c)}$
$\pi(\mathbf{n})$	Steady state probability of (aggregated) state $\mathbf{n}$ of a QN
Ω	Set of service centers of the QN, $\Omega = \{1, \dots, M\}$

and the capacity function. This can be a non-trivial task. A similar argument can be formulated for Proxy server station. On the other hand if we focus to BCMP model, we should really pay attention to the service time distribution only for those stations which have a FCFS discipline for which the only possible distribution is the exponential, while for the other types of nodes we can use Coxian distribution. This example and the QN model calibration are described in [52,39].

The network can be studied with the help of software tools. We can point out the bottlenecks by observing the utilization of the stations. For example, if the utilization of the incoming link is high ($U_7 \to 1$) one can decide either a new a agreement with the ISP in order to increase the incoming link bandwith, or to improve the Proxy cache hit rate.

References

1. Afshari, P. V., Bruell, S. C., and Kain, R. Y. Modeling a new technique for accessing shared buses. In *Proc. of the Computer Network Performance Symposium* (New York, NY, USA, 1982), ACM Press, pp. 4–13.
2. Akyildiz, I. F. Exact product form solution for queueing networks with blocking. *IEEE Trans. on Computer C-36-1* (1987), 122–125.
3. Akyildiz, I. F., and Bolch, G. Mean value analysis approximation for multiple server queueing networks. *Perform. Eval. 8*, 2 (1988), 77–91.
4. Baccelli, F., Massey, W., and Towsley, D. *Acyclic fork-join queueing networks*, vol. 36. 1989, pp. 615–642.
5. Balsamo, S., De Nitto Personé, V., and Onvural, R. *Analysis of Queueing Networks with Blocking*. Kluwer Academic Publishers, 2001.
6. Balsamo, S., and Iazeolla, G. An extension of Norton's theorem for queueing networks. *IEEE Trans. on Software Eng. SE-8* (1982).
7. Balsamo, S., and Iazeolla, G. Product-form synthesis of queueing networks. *IEEE Trans. on Software Eng. SE-11*, 2 (1985), 194–199.
8. Bard, Y. Some extensions to multiclass queueing network analysis. In *Proc. of the Third Int. Symp. on Modelling and Performance Evaluation of Computer Systems* (Amsterdam, NL, 1979), North-Holland Publishing Co., pp. 51–62.
9. Baskett, F., Chandy, K. M., Muntz, R. R., and Palacios, F. G. Open, closed, and mixed networks of queues with different classes of customers. *J. ACM 22*, 2 (1975), 248–260.
10. Bolch, G., Greiner, S., de Meer, H., and Trivedi, K. S. *Queueing networks and Markov chains*. John Wiley, 1998.
11. Boucherie, R., and van Dijk, N. M. Product-form queueing networks with state dependent multiple job transitions. *Advances in Applied Prob. 23* (1991), 152–187.
12. Bruell, S., and Balbo, G. *Computational Algorithms for Closed Queueing Networks*. The Computer Science Library. Elsevier North Holland, 1980.
13. Buzen, J. P. Computational algorithms for closed queueing networks with exponential servers. *Commun. ACM 16*, 9 (1973), 527–531.
14. Calzarossa, M., and Tucci, S., Eds. *Performance Evaluation of Complex Systems: Techniques and Tools, Performance 2002, Tutorial Lectures* (London, UK, 2002), Springer-Verlag.

15. Chandy, K., Herzog, U., and Woo, L. Parametric analysis of queueing networks. *IBM Journal of Res. and Dev. 1*, 1 (1975), 36–42.
16. Chandy, K. M., Hergox, U., and Woo, L. Approximate analysis of general queueing networks. *IBM Journal of Res. and Dev. 19* (1975), 43–49.
17. Chandy, K. M., John H. Howard, J., and Towsley, D. F. Product form and local balance in queueing networks. *J. ACM 24*, 2 (1977), 250–263.
18. Chandy, K. M., and Martin, A. J. A characterization of product-form queuing networks. *J. ACM 30*, 2 (1983), 286–299.
19. Chandy, K. M., and Neuse, D. Linearizer: a heuristic algorithm for queueing network models of computing systems. *Commun. ACM 25*, 2 (1982), 126–134.
20. Chandy, K. M., and Sauer, C. H. Computational algorithms for product form queueing networks. *Commun. ACM 23*, 10 (1980), 573–583.
21. Cohen, J. W. *The single server queue.* Wiley-Interscience, 1969.
22. Conway, A. E., de Souza e Silva, E., and Lavenberg, S. S. Mean Value Analysis by chain of product form queueing networks. *IEEE Trans. Comput. 38*, 3 (1989), 432–442.
23. Conway, A. E., and Georganas, N. D. Recal - a new efficient algorithm for the exact analysis of multiple-chain closed queuing networks. *J. ACM 33*, 4 (1986), 768–791.
24. Conway, A. E., and Georganas, N. D. *Queueing Networks - Exact Computational Algorithms: A unified Theory Based on Decomposition and Aggregation.* The MIT Press, Cambridge, MA, 1989.
25. Courtois, P. *Decomposability.* Academic Press, New York, 1977.
26. de Souza e Silva, E., and Lavenberg, S. S. Calculating joint queue-length distributions in product-form queuing networks. *J. ACM 36*, 1 (1989), 194–207.
27. Fourneau, J.-M., Gelenbe, E., and Suros, R. G-networks with multiple class negative and positive customers. In *MASCOTS '94: Proc. of the Second International Workshop on Modeling, Analysis, and Simulation On Computer and Telecommunication Systems* (Washington, DC, USA, 1994), IEEE Computer Society, pp. 30–34.
28. Fourneau, J.-M., and Verchere, D. G-networks with triggered batch state-dependent movement. In *Proc. MASCOTS* (1995), pp. 33–37.
29. Gelenbe, E. Product form networks with negative and positive customers. *Journal of Applied Prob. 28*, 3 (1991), 656–663.
30. Gelenbe, E. G-networks: a unifying model for neural and queueing networks. *Annals of Operations Research 48*, 5 (October, 1994), 433–461.
31. Gelenbe, E., and Fourneau, J.-M. G-networks with resets. *Perform. Eval. 49*, 1-4 (2002), 179–191.
32. Gelenbe, E., and Mitrani, I. *Analysis and Synthesis of Computer Systems.* Academic Press, New York, 1980.
33. Gordon, W. J., and Newell, G. F. Cyclic queueing networks with exponential servers. *Operations Research 15*, 2 (1967), 254–265.
34. Gordon, W. J., and Newell, G. F. Cyclic queueing networks with restricted length queues. *Operations Research 15*, 2 (1967), 266–277.
35. Henderson, W., and Taylor, P. Product form in networks of queues with batch arrivals and batch services. *Queueing Systems 6* (1990), 71–88.
36. Henderson, W., and Taylor, P. Some new results on queueing networks with batch movements. *Journal of Applied Prob. 28* (1990), 409–421.
37. Hoyme, K. P., Bruell, S. C., Afshari, P. V., and Kain, R. Y. A tree-structured Mean Value Analysis algorithm. *ACM Trans. Comput. Syst. 4*, 2 (1986), 178–185.

38. Jackson, J. Jobshop-like queueing systems. *Management Science 10* (1963), 131–142.
39. Kant, K. *Introduction to Computer System Performance Evaluation.* McGraw-Hill, 1992.
40. Kelly, F. *Reversibility and stochastic networks.* Wiley, New York, 1979.
41. Kemeny, J. G., and Snell, J. L. *Finite Markov Chains.* D. Van Nostrand Company, inc., 1960, ch. II.
42. Kleinrock, L. *Queueing Systems*, vol. 1 (Theory). John Wiley and Sons, 1975.
43. Kritzinger, P., van Wyk, S., and Krezesinski, A. A generalization of Norton's theorem for multiclass queueing networks. *Performance Evaluation 2* (1982), 98–107.
44. Lam, S. S. Queueing networks with capacity constraints. *IBM Journal of Res. and Dev. 21*, 4 (1977), 370–378.
45. Lam, S. S. Dynamic scaling and growth behavior of queuing network normalization constants. *J. ACM 29*, 2 (1982), 492–513.
46. Lam, S. S., and Lien, Y. L. A tree-convolution algorithm for the solution of queueing networks. *Commun. ACM 26*, 3 (1983), 203–215.
47. Lavenberg, S., and Reiser, M. Stationary state probabilities at arrival instants for closed queueing networks with multiple types of customers. *Journal of Applied Prob. 17* (1980), 1048–1061.
48. Lavenberg, S. S. *Computer Performance Modeling Handbook.* Academic Press, New York, 1983.
49. Lazowska, E. D., Zahorjan, J. L., Graham, G. S., and Sevcick, K. C. *Quantitative system performance: computer system analysis using queueing network models.* Prentice Hall, Englewood Cliffs, NJ, 1984.
50. Le Boudec, J.-Y. A BCMP extension to multiserver stations with concurrent classes of customers. In *SIGMETRICS '86/PERFORMANCE '86: Proc. of the 1986 ACM SIGMETRICS Int. Conf. on Computer performance modelling, measurement and evaluation* (New York, NY, 1986), ACM Press, pp. 78–91.
51. Marie, R. An approximate analytical method for general queueing networks. *IEEE Trans. on Software Eng. 5*, 5 (1979), 530–538.
52. Menascè, D. A., and Almeida, V. A. F. *Capacity Planning for Web Performance. Metrics, Models, & Methods.* Prentice Hall, 1998.
53. Muntz, R. Network of queues. Tech. Rep. Notes for Engineering 226 C., University of Los Angeles, Department of Computer Science, 1972.
54. Muntz, R., and Wong, J. Efficient computational procedures for closed queueing network models. In *Proc. 7th Hawaii Int. Conf. on System Science* (January 1974), pp. 33–36.
55. Muntz, R. R. Poisson departure processes and queueing networks. Tech. Rep. IBM Research Report RC4145, Yorktown Heights, New York, 1972.
56. Nelson, R. The mathematics of product-form queueing networks. *ACM Computing Survey 25*, 3 (1993), 339–369.
57. Neuse, D., and Chandy, K. SCAT: A heuristic algorithm for queueing network models of computing systems. In *Proc. of ACM SIGMETRICS Conf. on measurement and modeling of computer systems* (New York, NY, 1981), ACM Press, pp. 59–79.
58. Neuts, M. F. *Matrix Geometric Solutions in Stochastic Models.* John Hopkins, Baltimore, Md, 1981.
59. Noetzel, A. S. A generalized queueing discipline for product form network solutions. *J. ACM 26*, 4 (1979), 779–793.

60. Pattipati, K. R., Kostreva, M. M., and Teele, J. L. Approximate mean value analysis algorithms for queuing networks: existence, uniqueness, and convergence results. *J. ACM 37*, 3 (1990), 643–673.
61. Petriu, D. C., Neilson, J. E., Woodside, C. M., and Majumdar, S. Software bottlenecking in client-server systems and rendezvous networks. *IEEE Trans. on Software Eng. 21*, 9 (1995), 776–782.
62. Raiser, M. Mean Value Analysis and Convolution method for queue-dependent servers in closed queueing networks. *Performance Evaluation 1*, 1 (1981), 7–18.
63. Reiser, M., and Sauer, C. H. Queueing network models: Methods of solution and their program implementation. In *Current Trends in Programming Methodology*, (K. M. Chandy and R. T. Yeh), Ed. Prentice-Hall Inc., 1978.
64. Resiser, M., and Lavenberg, S. S. Mean Value Analysis of closed multichain queueing network. *J. ACM 27*, 2 (1980), 313–320.
65. Rolia, J., and Sevcick, K. The methods of layers. *IEEE Trans. on Software Eng. 21*, 8 (1995), 682–688.
66. Sahner, R., Trivedi, K., and Puliafito, A. *Performance and Reliability Analysis of Computer Systems - An Example-Based Approach Using the SHARPE Software Package.* Kluwer Academic Publishers, 1996.
67. Sauer, C. H. Computational algorithms for state-dependent queueing networks. *ACM Trans. Comput. Syst. 1*, 1 (1983), 67–92.
68. Sauer, C. H., and Chandy, K. M. *Computer Systems performance modeling.* Prentice-Hall, Englewood Cliffs, 1981.
69. Schweitzer, P. Approximate analysis of multiclass closed network of queues. In *Proc. of Int. Conf. on Stochastic Control and Optimization* (Amsterdam, NL, 1979).
70. Sevcik, K. C., and Mitrani, I. The distribution of queuing network states at input and output instants. *J. ACM 28*, 2 (1981), 358–371.
71. Shassenberger, R. The insensitivity of stationary probabilities in networks of queues. *Journal of Applied Prob. 10* (1978), 85–93.
72. Smith, C. *Performance Engineering of Software Systems.* Addison-Wesley, 1990.
73. Strelen, J. A generalization of Mean Value Analysis to higher moments: moment analysis. In *SIGMETRICS '86/PERFORMANCE '86: Proc. of 1986 ACM SIGMETRICS Int. Conf. on computer performance modelling, measurement and evaluation* (New York, NY, 1986), ACM Press, pp. 129–140.
74. Suri, R. Robustness of queuing network formulas. *J. ACM 30*, 3 (1983), 564–594.
75. Towsley, D. Queuing network models with state-dependent routing. *J. ACM 27*, 2 (1980), 323–337.
76. Trivedi, K. S. *Probability and statistics with reliability, queuing and computer science applications*, second ed. Wiley-Interscience, 2002.
77. Tucci, S., and Sauer, C. The tree MVA algorithm. *Performance Evaluation*, 5(3) (August 1985), 187–196.
78. van Dijk, N. *Queueing networks and product forms.* John Wiley, 1993.
79. Whittle, P. Partial balance and insensitivity. *Journal of Applied Prob. 22* (1985), 168–175.

Introduction to Generalized Stochastic Petri Nets

Gianfranco Balbo

Università di Torino,
Dipartimento di Informatica
Corso Svizzera, 185, 10149 Torino, Italy
balbo@di.unito.it

Abstract. Generalized Stochastic Petri Nets are a modelling formalism that can be conveniently used for the analysis of complex models of Discrete Event Dynamic Systems and for their performance and reliability evaluation. The automatic construction of the probabilistic models that underly the dynamic behaviours of these nets rely on a set of results that derive from the theory of untimed Petri nets. The paper briefly surveys some results of net theory together with the different approaches used to introduce the concept of time in these models that are useful for the definition of Stochastic Petri Nets and Generalized Stochastic Petri Nets. Details on the solution techniques and on their computational aspects are provided. A brief overview of advanced material is included at the end of the paper to highlight the state of the art in this field and to give pointers to relevant results published in the literature.

1 Introduction

Petri nets [47,1,46,48] are a powerful tool for the description and the analysis of systems that exhibit concurrency, synchronization and conflicts. Timed Petri nets [7,41] in which the basic model is augmented with time specifications are commonly used to evaluate the performance and reliability of complex systems.

The pioneering work in the area of timed Petri nets was performed by Noe and Nutt [45] and by Merlin and Faber [38]. In this early work, timed Petri nets were viewed as a formalism for the description of the global behaviour of complex structures. The nets were used to drive simulations so that their analysis was conducted on the basis of observations made during the different runs.

Following these initial ideas, several proposals for incorporating timing information into Petri net models appeared in the literature. Interpreting Petri nets as state/event models, time is naturally associated with activities that induce state changes, and hence with the delays incurred before firing transitions.

Stochastic Petri Nets (SPNs) were introduced in 1980 [49,43,40] as a formalism for the description of Discrete Event Dynamic Systems (DEDS) whose dynamic behaviour could be represented by means of continuous-time homogeneous Markov chains. The original SPN proposal assumed atomic firings, exponentially distributed firing times, and a race execution policy.

M. Bernardo and J. Hillston (Eds.): SFM 2007, LNCS 4486, pp. 83–131, 2007.

With the aim of extending the modelling power of stochastic Petri nets, Generalized Stochastic Petri Nets (GSPNs) were proposed in [4]. GSPNs include two classes of transitions: exponentially distributed *timed* transitions, which are used to model the random delays associated with the execution of activities, and *immediate* transitions, which are devoted to the representation of logical actions that do not consume time. Immediate transitions permit the introduction of branching probabilities that are independent of timing specifications. When timed and immediate transitions are enabled in the same marking, immediate transitions always fire first. In GSPNs the reachability set is also partitioned in two sets. *Tangible* markings are those in which only timed transitions are enabled whereas *vanishing* markings are those in which at least one immediate transition is enabled. The time spent by a GSPN in a tangible state is exponentially distributed with the parameter depending on the timed transitions that are enabled in that marking; the time spent by a GSPN in a vanishing marking is instead zero. Other generalizations of the basic SPN formalism that are related to GSPNs, are the Extended Stochastic Petri Nets [29], the Stochastic Activity Networks [39], and the Well Formed Stochastic Petri Nets that allows transitions and tokens to be coloured [19]. GSPNs are among the SPN formalisms that are most commonly used for the analysis of important problems and a considerable effort has been devoted to their improvement since the time of their original introduction.

To overcome the constraints introduced by the exponential distribution of firing of timed transitions, several further extentions have been introduced that allow the construction of more complex underlying stochastic processes. Crucial in the definition of these models is the characterization of the behaviour of concurrent time transitions when one of them fires, as well as that of conflicting transitions that may become disabled at the solution of the conflict. Special classes of GSPNs with non-exponential firing time distributions are the Deterministic Stochastic Petri Nets (DSPN) that account for the presence of deterministic distributions and the Phase Type Stochastic Petri Nets (PHSPN)in which the non-exponential distributions are limited to be of Phase Type.

In this paper, we discuss the relevance of Generalized Stochastic Petri Nets by providing first an introduction to the basic results that set the ground for the derivation of the stochastic processes corresponding to these models and for the study of their solution methods. The balance of the paper is the following. Section 2 describes the relevance of Petri nets for modelling Discrete Event Dynamic Systems and introduces the basic classical properties of the formalism that are needed later in the paper. Section 3 discusses the impact that transitions of different priority levels have on the analysis of the model. Section 4 presents the different possibilities that exist for introducing the concept of time in Petri net models. Section 5 introduces the definition of Stochastic Petri nets and provides the details for the construction of their underlying stochastic process. Section 6 discusses the characteristics of Generalized Stochastic Petri Nets and provides details on some of the computational issues that are relevant for the application of this modeling formalism. Section 7 concludes the paper with a few pointers to

more advanced material. and with some general remarks on net modelling. The paper has been written in the attempt of providing a uniform and self-contained introduction to modelling with Generalized Stochastic Petri Nets. The discussion introduces the basic terminology and operational rules of Petri nets for which a comprehensive reference can be found in [42] where the reader will be able to find a clear explanation of all the concepts that are only marginally addressed in this work. Moreover, additional explanations of the topics discussed throughout the paper can be found in [6,11,12,13].

2 Petri Net

Petri nets (PNs) are abstract formal models that have been developed in search for natural, simple, and powerful methods for describing and analyzing the flow of information and control in systems.

PNs have been originally proposed for the description and the analysis of systems in which concurrency and conflicts play a special role. In PNs, the state of the system derives from the combination of local state variables that allow a direct representation of concurrency, causality, and independence. PNs are bipartite graphs represented as collections of places and transitions connected by directed arcs. The graphical aspect of these models is very attractive for practical modelling, since it helps in understanding how features of the real system are conveyed in the model.

In order to keep PN models concise, high level PN have been introduced that provide a form of abbreviation when repetition of similar sub-nets would make the model large and difficult to understand. Always to make models easier to understand, several extensions have been introduced in the basic PN formalism, often with the disadvantage of reducing the *analyzability* of the model (e.g., inhibitor arcs give to the Petri net formalism the computation power of Turing Machines, but their effect cannot be accounted for using the standard methods for the structural analysis of the net).

The behaviour of PNs is independent of time and environment, and is characterized by the *non-deterministic* firing of transitions that are simultaneously enabled in a given marking. The connection of the formalism with reality is provided in this case by *interpretations* that incorporate in the model external constraints such as time considerations. Different extensions and different interpretations yield different PN based formalisms sharing some basic principles.

2.1 Petri Net Models

PN models consist of two parts: a net structure and a marking. The net structure is an inscribed bipartite directed graph, that represents the static part of the system. The two types of nodes, places and transitions, are represented as circles and boxes (or bars), respectively. Places correspond to state variables of the system and transitions to actions that induce changes of states. Arcs connecting places to transitions are called *input* arcs; *output* arcs connect instead transitions

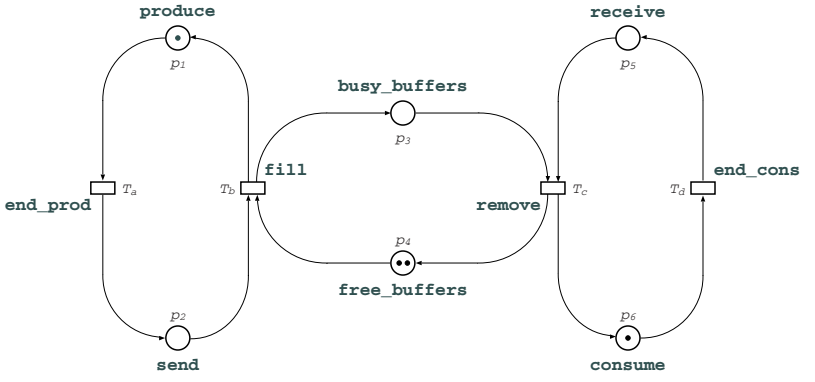


Fig. 1. Simple PN model of the classical Consumer/Producer problem

to places. Different types of inscriptions lead to various families of nets. When the inscriptions are natural numbers associated with arcs, named weights or multiplicities, Place/Transition (P/T) nets are obtained.

The marking is an assignment of tokens to places. The marking of a place represents its state value.

The specification of a PN model thus requires the definition of the net and the assignment of the initial marking. The dynamics of a system (i.e., its behaviour) is given by the evolution of the marking that is driven by few simple rules:

- A transition occurs when the input state values fulfill some conditions expressed by the arc inscriptions.
- The occurrence of a transition changes the values of its adjacent state variables (markings of input and output places) according to arc inscriptions.

Figure 1 illustrates this graphically capability by modeling a pair of Producer-/Consumer processes communicating via an intermediate buffer. The Producer process is characterized by two phases: production and delivery represented by the transitions *end_prod* and *fill*. The delivering of a product can be done only if the buffer has (at least) one position available. Free buffer positions are counted by the tokens in place *free_buffers*. If no free buffer positions are available, the Producer waits in place *send* for one position to become free. Similarly, the Consumer process, waits for a product to be available in the buffer. When the product arrives the Consumer removes it from the buffer, thus making the position available for future deliveries, and enters a phase of local processing (the token in place *consume*) that, when completed, makes the Consumer ready for accepting the next product. This separation between the structure of the net and its dynamics allows to reason on net based models at two different levels: *structural* and *behavioural*. From the former we may derive "fast" conclusions on the possible behaviours of the modelled system. Pure behavioural reasonings can be more conclusive, but they may require substantial computations, which in certain cases may not even be feasible. Structural reasoning may be regarded as

an *abstraction* of the behavioural one: for instance, instead of studying whether a given system has a finite state space, we might address the problem of whether the state space is finite *for every* possible initial marking; similarly, we could investigate whether *there exists* an initial marking that guarantees infinite activity, rather than verifying if this is the case for a given initial state.

A marked PN is formally defined by the following tuple

$$PN \;=\; (P,\; T,\; F,\; W,\; \boldsymbol{m}_0) \tag{1}$$

where
$P \;=\; (p_1,\; p_2,\; ...,\; p_P)$ is the set of places,
$T \;=\; (t_1,\; t_2,\; ...,\; t_T)$ is the set of transitions,
$F \;\subseteq\; (P \times T) \bigcup (T \times P)$ is the set of arcs,
$W \;:\; F \;\rightarrow \mathbb{N}$ is a weight function,
$\boldsymbol{m}_0 \;=\; (m_{01},\; m_{02},\; ...,\; m_{0P})$ is the initial marking.

As we have said, a marking $\boldsymbol{m}$ is an assignment of tokens to places and can thus be represented by a vector with as many components as there are places in the net: the $i-th$ component representing the number of tokens assigned to place p_i. The dot notation is used for pre- and post-sets of nodes: ${}^\bullet v = \{u| < u, v > \in F\}$ and $v^\bullet = \{u| < v, u > \in F\}$. A pair comprising a place p and a transition t is called a *self-loop* if p is both input and output of t ($p \in {}^\bullet t \;\bigwedge\; p \in t^\bullet$). A net is said to be pure if it has no self-loops. Pure nets are completely characterised by a single matrix $\boldsymbol{C}$ that is called the *incidence matrix* of the net and that combines the information provided by the flow relations and by the weight function.

$$\boldsymbol{C} \;=\; \begin{array}{c} \\ \text{places} \\ \end{array} \begin{array}{c} \overset{\textit{transitions}}{\overbrace{\qquad\qquad}} \\ \boxed{c_{pt}} \end{array}$$

with $\quad c_{pt} \;=\; c^+_{pt} + c^-_{pt} \;=\; w(t,p) \;-\; w(p,t).$

Table 1 contains the specifications of these formal components for the model of Fig. 1.

2.2 System Dynamics

The graph and matrix characterizations that we have introduced in the previous section represent the static component of a PN model. The dynamic evolution of the PN marking is governed by transition occurrences (firings) which consume and create tokens.

"Enabling" and "firing" rules are associated with transitions. Informally, we can say that the enabling rule defines the *conditions* that allow a transition to fire, and the firing rule specifies the *change* of state produced by the transition. Both the enabling and the firing rules are specified in terms of arc characteristics.

Table 1. Formal specification of the Producer/Consumer Petri Net model of Fig. 1

Set of places:	$P = (p_1,\ p_2,\ p_3,\ p_4,\ p_5,\ p_6)$
Set of transitions:	$T = (a,\ b,\ c,\ d)$
Incidence matrix:	$\mathbf{C} =$

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \left[\begin{array}{cccc} -1 & +1 & & \\ +1 & -1 & & \\ & +1 & -1 & \\ & -1 & +1 & \\ & & -1 & +1 \\ & & +1 & -1 \end{array}\right] \end{array}$$

Initial marking:	$\boldsymbol{m}_0 = (1,\ 0,\ 0,\ 2,\ 0,\ 1)$

A transition t is *enabled* if and only if each input place contains a number of tokens *greater or equal* than given thresholds defined by the multiplicities of arcs. The set of transitions enabled in marking $\boldsymbol{m}$ is indicated with $E(\boldsymbol{m})$; the number of simultaneous enablings of a transition t_i in a given marking $\boldsymbol{m}$ is called its enabling degree, and is denoted by $e_i(\boldsymbol{m})$.

When transition t fires, it *deletes* from each place in its input set ${}^\bullet t$ as many tokens as the multiplicity of the arc connecting that place to t, and *adds* to each place in its output set $t^\bullet$ as many tokens as the multiplicity of the arc connecting t to that place.

Transition t, enabled in marking $\boldsymbol{m}$, fires producing marking $\boldsymbol{m}'$ such that $\boldsymbol{m}' = \boldsymbol{m} + O(t) - I(t)$. The change of state due to the firing of transition t is usually indicated in a compact way as $\boldsymbol{m}[t\rangle\boldsymbol{m}'$, and we say that $\boldsymbol{m}'$ is *directly reachable* from $\boldsymbol{m}$.

The natural extension of the concept of transition firing, is the firing of a *transition sequence* (or execution sequence). A transition sequence[1] $\sigma = t_{(1)}, \cdots, t_{(k)}$ can fire starting from marking $\boldsymbol{m}$ if and only if there exists a *marking sequence*[2] $\Sigma = \boldsymbol{m}_{(1)}, \boldsymbol{m}_{(2)}, \cdots, \boldsymbol{m}_{(k+1)}$ with $\boldsymbol{m} = \boldsymbol{m}_{(1)}$ and $\boldsymbol{m}' = \boldsymbol{m}_{(k+1)}$, such that $\forall i = (1, \cdots, k), \boldsymbol{m}_{(i)}[t_{(i)}\rangle\boldsymbol{m}_{(i+1)}$. We denote by $\boldsymbol{m}[\sigma\rangle\boldsymbol{m}'$ the firing of a transition sequence, and we say that $\boldsymbol{m}'$ is *reachable* from $\boldsymbol{m}$.

An important final consideration is that the enabling and firing rules for a generic transition t are "local": indeed, only the numbers of tokens in the input of t, and the weights of the arcs connected to t need to be considered to establish

[1] We write $t_{(1)}$ rather than t_1 because we want to indicate the first transition in the sequence, that may not be the one named t_1.

[2] Also in this case $\boldsymbol{m}_{(1)}$ represents the first marking of the sequence rather than a specific marking named $\boldsymbol{m}_1$.

whether t can fire and to compute the change of marking induced by its firing. This justifies the assertion that the PN marking is intrinsically "distributed".

A common way of describing the behaviour of a PN is by means of its sequential observation. A hypothetical observer is supposed to "see" only single events occurring at any point in time. The interleaving semantics of a net system is given by all possible sequences of individual transition firings that could be observed from the initial marking. If two transitions t_1 and t_2 are enabled simultaneously, and the occurrence of one does not disable the other, in principle they could occur at the same time, but the sequential observer will see either t_1 followed by t_2 or viceversa. The name interleaving semantics comes from this way of "seing" simultaneous occurrences.

An alternative way of envisioning the evolution of a net is that of allowing the presence of *multiple events*. Multiple events may happen at any given time. A *step* S is a multi-set of transitions that are enabled to concurrently fire in the same marking. Firing a step (*step semantics*) amounts to withdraw the tokens from all the input places of the transitions of the step and to deposit tokens in all their output places. If a set of transitions can be fired in a step, this makes explicit the fact that they need not occurring in a precise order. The occurrence of a step can be denoted by $\boldsymbol{m} \xrightarrow{S} \boldsymbol{m}'$, or $\boldsymbol{m} \xrightarrow{\sigma} \boldsymbol{m}'$, if σ is an arbitrary sequentialisation of S. In fact, every sequentialisation of the step is fireable so that, in practice, the reachable markings can be computed considering individual transition occurrences only.

Starting from the initial marking it is possible to compute the set of all markings reachable from it (the state space of the PN) and all the paths that the system may follow to move from state to state.[3] When there is no possibility of confusion, we indicate with RS the set $RS(\boldsymbol{m}_0)$. We also indicate with $RS(\boldsymbol{m})$ the set of markings reachable from a generic marking $\boldsymbol{m}$. A marking $\boldsymbol{m}'$ is said to be a *home state* iff it can be reached from any $\boldsymbol{m} \in RS(\boldsymbol{m}_0)$. The RS contains no information about the transition sequences fired to reach each marking.

This information is contained in the reachability graph (RG), where each node represents a reachable state, and there is an arc from $\boldsymbol{m}_1$ to $\boldsymbol{m}_2$ if the marking $\boldsymbol{m}_2$ is directly reachable from $\boldsymbol{m}_1$. If $\boldsymbol{m}_1[t\rangle\boldsymbol{m}_2$, the arc is labelled with t. Note that more than one arc can connect two nodes (it is indeed possible for two transitions to be enabled in the same marking and to produce the same state change), so that the reachability graph is actually a multigraph.

For our practical example using the Producer/Consumer problem, the RS is listed in Table 2, while the RG is depicted in Fig. 2.

The dynamic behaviour of PN models is characterized by three basic phenomena that account for the fact that actions may occur simultaneously (*concurrency*), some require that others occur first (*causal dependency*), and they may occur only in alternative (*conflicts*).

[3] Obviously this computation is feasible only in the case of models with finite state spaces. In the rest of this paper, we assume that our models satisfy this condition, except when it is stated differently. Proper generalisations are possible to deal with infinite state spaces introducing the notion of "covering tree" [42].

Table 2. Reachability Set of the Producer/Consumer Petri Net model of Fig. 1

$\boldsymbol{m}_0 = (1, 0, 0, 2, 0, 1)$	$\boldsymbol{m}_1 = (0, 1, 0, 2, 0, 1)$
$\boldsymbol{m}_2 = (1, 0, 0, 2, 1, 0)$	$\boldsymbol{m}_3 = (1, 0, 1, 1, 0, 1)$
$\boldsymbol{m}_4 = (0, 1, 0, 2, 1, 0)$	$\boldsymbol{m}_5 = (0, 1, 1, 1, 0, 1)$
$\boldsymbol{m}_6 = (1, 0, 1, 1, 1, 0)$	$\boldsymbol{m}_7 = (1, 0, 2, 0, 0, 1)$
$\boldsymbol{m}_8 = (0, 1, 1, 1, 1, 0)$	$\boldsymbol{m}_9 = (0, 1, 2, 0, 0, 1)$
$\boldsymbol{m}_{10} = (1, 0, 2, 0, 1, 0)$	$\boldsymbol{m}_{11} = (0, 1, 2, 0, 1, 0)$

Concurrency - Two transitions are concurrent in a given marking if they can occur in a step. Transitions t_i and t_j are concurrent in marking $\boldsymbol{m}$, if $\boldsymbol{m} > \boldsymbol{c}(., t_i)^T + \boldsymbol{c}(., t_j)^T$. Notice that steps allow to express true concurrency. In the case of interleaving semantics, concurrency of two (or more) actions t_1 and t_2 is represented by the possibility of performing them in any order, first t_1 and then t_2, or viceversa. Nevertheless, the presence of all possible sequentialisations of the actions does not imply that they are "truly" concurrent, as the example in Figure 3 illustrates: t_1 and t_2 can occur in any order, but they cannot occur simultaneously, and in fact the step $t_1 + t_2$ is not enabled. The distinction is

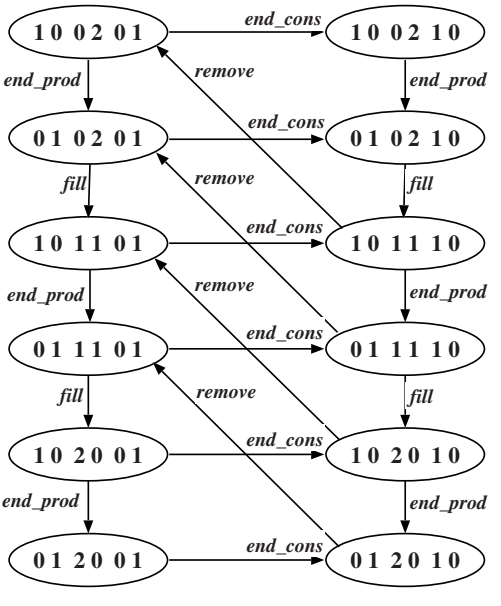


Fig. 2. Reachability Graph of the PN of Fig. 1

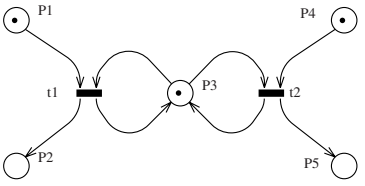


Fig. 3. Shared place

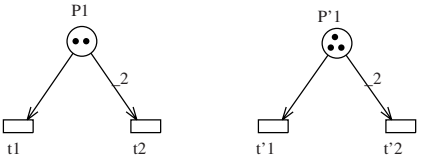


Fig. 4. Conflict situations

especially important if transitions t_1 and t_2 were to be refined, i.e., if they were to be replaced by subnets.

Causal dependence - Informally, causal dependencies are represented by the partial ordering of actions induced by the flow relation. They correspond to situations in which the firing of a given transition can happen only after the occurrence of others in whatever order. The very basic net construct used to model causal dependence is a place connecting two transitions.

Conflicts - Informally, we have a situation of *conflict* when, being several transitions enabled in the same marking, we must chose which one to fire and, by so doing, we affect the enabling conditions of the others. The very basic net construct used to model conflicts is a place with more than one output transition. We can say that a transition t_r is in conflict with transition t_s in marking $\boldsymbol{m}$ iff $t_r, t_s \in E(\boldsymbol{m})$, $\boldsymbol{m} \xrightarrow{t_s} \boldsymbol{m}'$, and $t_r \notin E(\boldsymbol{m}')$. Things are more complex when we consider concurrent systems where the fact that two transitions are enabled in a given marking does not necessarily means that we have to choose which one to fire even if they share some input places. Conflicts can be antisymmetric as we can see from the second example of Figure 4 where t'_2 is in conflict with t'_1, but not the other way around, since the firing of t'_1 does not decrease the enabling degree of t'_2.

Conflicts are called *equal* when all the transitions have the same input set.

An intriguing situation arises when different interleaved firings of the members of a step may either yield conflict situations or not. This phenomenon is known as *confusion* and is illustrated by the conflicts of Figure 5, where we can recognize that t_i and t_k are a step such that, when t_k fires no effects are felt by transition t_i, while the same is not true for transition t_j.

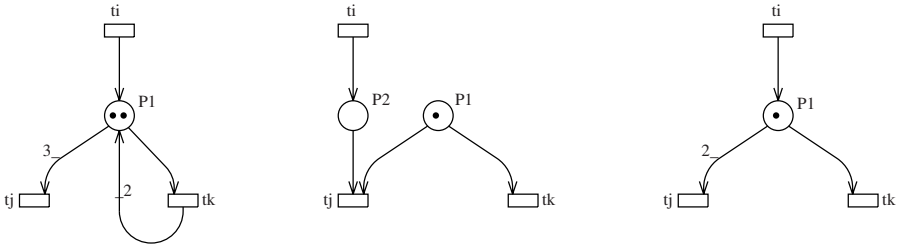


Fig. 5. Non equal conflicts

These few examples show how the concept of conflict in PNs is intrinsically complex and crucial for their analysis. A comprehensive discussion of these aspects is contained in [6,13] where the interested reader can find a formal definition of the properties mentioned so far as well as additional examples useful for their clarifications.

2.3 Properties of Petri Nets

Properties of PN models are characteristics that allow to assess the quality of a given system in an objective manner. The following are among the most useful properties that can be defined for PN models.

Reachability and reversibility — As defined before, a marking $\boldsymbol{m}'$ is *reachable* from $\boldsymbol{m}$ if there exists a sequence σ such that $\boldsymbol{m}[\sigma\rangle\boldsymbol{m}'$.

Reachability can be used to answer questions concerning the possibility for the modelled system of being in a given marking $\boldsymbol{m}$. An important reachability property is *reversibility*: a marked PN is said to be *reversible* if and only if from any state reachable from $\boldsymbol{m}_0$, it is possible to come back to $\boldsymbol{m}_0$ itself. Reversibility expresses the possibility for a PN to come back infinitely often to its initial marking.

Liveness — A transition t is said to be *live* if and only if, for each marking $\boldsymbol{m}$ reachable from $\boldsymbol{m}_0$, there exists a marking $\boldsymbol{m}'$, reachable from $\boldsymbol{m}$, such that $t \in E(\boldsymbol{m}')$. A PN is said to be live iff $\forall t_r \in T : t_r$ is live. Liveness is a property that depends on the initial marking. A transition that is not live is said to be *dead*. For each dead transition t, it is possible to find a marking $\boldsymbol{m}$ such that none of the markings in $RS(\boldsymbol{m})$ enables t.

A very important consequence of liveness is that, if at least one transition is live, then the PN cannot deadlock.[4] Moreover, if all transitions are live, then the corresponding PN contains no livelock.[5] Liveness defines the possibility for a transition to be enabled (and to fire) infinitely often.

[4] A PN contains a deadlock if it can reach a state in which no transition can be fired.
[5] A system is in a livelock condition when it enters a subset of its activities from which it has no possibility of exiting.

Boundedness — A place p of a PN is said to be *k-bounded* if and only if, for each reachable marking $\boldsymbol{m}$, the number of tokens in that place is less than or equal to k. A PN is said to be *k-bounded* if and only if all places $p \in P$ are *k-bounded*. PNs that are 1-bounded are said to be *safe*. A very important consequence of boundedness is that it implies the finiteness of the state space. In particular, if a PN comprising N places is *k-bounded*, the number of states cannot exceed $(k+1)^N$.

Being able to identify the properties of a PN model is an important step of its analysis that can be supported by the use of linear algebraic techniques, derive some basic properties of the net from the incidence matrix $\boldsymbol{C}$.

The relevance of the incidence matrix is due to the fact that it allows the net dynamics to be expressed by means of linear algebraic equations. In particular, we can observe that *for any marking* $\boldsymbol{m}$, the firing of a transition t enabled in $\boldsymbol{m}$ produces the new marking

$$\boldsymbol{m}' = \boldsymbol{m} + \boldsymbol{c}(.,t)^{\mathrm{T}} \tag{2}$$

where $\boldsymbol{m}$ and $\boldsymbol{m}'$ are row vectors, and $\boldsymbol{c}(.,t)$ is the column vector of $\boldsymbol{C}$ corresponding to transition t.

P-semiflows and P-invariant relations — A PN is *strictly conservative* (or strictly invariant) if and only if the token's count is constant for all the markings of the net, i.e. iff

$$\sum_{p=1}^{P} m_p \;=\; \sum_{p=1}^{P} m_{0p}, \qquad \forall \boldsymbol{m} \in RS(\boldsymbol{m}_0) \tag{3}$$

A PN is *conservative* (or P invariant) iff

$$\exists\, \boldsymbol{y} \;=\; (y_1,\ y_2,\ ...,\ y_P) > 0 \ \textit{such that}$$

$$\sum_{p=1}^{P} y_p m_p \;=\; \sum_{p=1}^{P} y_p m_{0p} \qquad \forall \boldsymbol{m} \in RS(\boldsymbol{m}_0) \tag{4}$$

i.e., the weighted token count is *invariant* with respect to any marking of the net. Taking into account that the incidence matrix $\boldsymbol{C}$captures the way in which markings change on the occurrence of transitions firings, it is easy to show that the vector $\boldsymbol{y}$ must satisfy the following equation:

$$\boldsymbol{C}^{\mathrm{T}} \cdot \boldsymbol{y} = \boldsymbol{0} \tag{5}$$

The positive vectors $\boldsymbol{y}$ that satisfy Equation (5) are called the *P-semiflows* of the PN. Note that P-semiflows are computed from the incidence matrix, and are thus independent of any notion of initial marking. Markings are only instrumental for their interpretation. In particular, the expression

$$\forall t \in T: \quad \sum_{p_i \in P} C(p_i, t) \cdot y_i = 0 \tag{6}$$

identifies an invariant relation called a place invariant, or simply *P-invariant*.

As a consequence, if in a PN model all places are covered by P-semiflows[6], then for any reachable marking (and independently of the initial marking), the maximum number of tokens in any place is finite (since the initial marking is finite) and the net is said to be *structurally bounded*.

All P-semiflows of a PN can be obtained as linear combinations of the P-semiflows that are elements of a minimal set PS. See [34,37,9,10] for P-semiflows computation algorithms.

T-semiflows and T-invariant relations — Using similar arguments based on the "cumulative change" of a marking produced by the execution of a transition sequence, it is possible to show that the solutions of the following equation have particular meanings as well:

$$\boldsymbol{C} \cdot \boldsymbol{x} = \boldsymbol{0} \tag{7}$$

The vectors $\boldsymbol{x}$, that are integer solutions of this matrix equation are called T-semiflows of the net.

In general, the invariant relation (called transition invariant or *T-invariant*) produced by a T-semiflow is the following:

$$\forall p \in P : \quad \sum_{t \in T} C(p,t) \cdot x(t) = 0 \tag{8}$$

This invariant relation states that, by firing from marking $\boldsymbol{m}$ any transition sequence σ whose transition count vector is a T-semiflow, the marking obtained at the end of the transition sequence is equal to the starting one, provided that σ can actually be fired from marking $\boldsymbol{m}$ ($\boldsymbol{m}[\sigma\rangle\boldsymbol{m}$). A net covered by T semiflows may have home states. A net with home states is covered by T-semiflows. Like P-semiflows, all T-semiflows can be obtained as linear combinations of the elements of a minimal set TS.

Note again that the T-semiflows computation is independent of any notion of marking, so that T-semiflows are identical for all PN models with the same structure and different initial markings.

Observe the intrinsic difference between P- and T-semiflows. The fact that all places in a PN are covered by P-semiflows is a sufficient condition for boundedness, whereas the existence of T-semiflows is only a necessary condition for a PN model to be able to return to a starting state, because there is no guarantee that a transition sequence with transition count vector equal to the T-semiflow can actually be fired.

Properties of PNs that are obtained from the incidence matrix and from the graph structure of the model, independently of the initial marking are identified as *structurals*.

Properties of PNs that depend on the initial marking and are obtained from the reachability graph (finite case) of the net or from the coverability tree (infinite case) are identified as *behaviourals*.

[6] A place p is covered by a P-semiflow if there is at least one vector $\boldsymbol{y}$ with a non null entry for p.

Applying these solution techniques to our practical example we obtain the results summarized in Table 3.

Table 3. Semiflows and invariant properties of the Producer/Consumer Petri Net model of Fig. 1

P-semiflows $(YC = 0)$:

$$y = (1,\ 1,\ 0,\ 0,\ 0,\ 0)$$
$$y = (0,\ 0,\ 1,\ 1,\ 0,\ 0)$$
$$y = (0,\ 0,\ 0,\ 0,\ 1,\ 1)$$

The net is covered by these P-semiflows and it is thus bounded.

T-semiflows $(CX = 0)$:

$$x = (1,\ 1,\ 1,\ 1)$$

The net is covered by this T-semiflow and this suggests the presence of a home state.

3 Petri Nets with Priorities and Inhibitor Arcs

In many practical situations it is useful having the possibility of assigning different levels of priority to the transitions of a PN. Similarly it can be very important having the capability of disabling the firing of a transition when a certain condition holds (and thus the corresponding place is marked). In this paper, we assume that the transitions satisfy a priority structure given once for all at the moment of the definition of the net so that in each marking, the choice of the transition to fire when several transitions are enabled is decided on the basis of this additional "static" information.

A marked PN with transition priorities, arc multiplicities, and inhibitor arcs can be formally defined by the following tuple:

$$PN = (P,\ T,\ \Pi(.),\ I(.),\ O(.),\ H(.),\ \boldsymbol{m}_0) \tag{9}$$

- P is a set of places,
- T is a set of transitions,
- $\boldsymbol{m}_0$ is an initial marking,
- $\Pi(.)$, $I(.)$, $O(.)$, $H(.)$ are four functions defined on T.

The priority function $\Pi(.)$ maps transitions into non-negative natural numbers representing their priority level. The input, output, and inhibition functions $I(.)$, $O(.)$, and $H(.)$ map transitions on "bags" of places. The former two are represented as directed arcs from places to transitions and viceversa; the inhibition function is represented by circle-headed arcs. When greater than one, the multiplicity is written as a number next to the corresponding arc.

As we just said, the priority definition that we assume in this paper is global: the enabled transitions with a given priority k always fire before any other enabled transition with priority[7] $j < k$.

This kind of priority definition can be used for two different modelling purposes: (1) it allows to partition the transition set into classes representing actions at different logical levels, e.g. actions that take time versus actions corresponding to logical choices that occur instantaneously; (2) it gives the possibility of specifying a deterministic conflict resolution criterion.

Enabling and firing — The firing rule in PNs with priority requires the following new definitions:

- a transition t_j is said to *have concession* in marking $\boldsymbol{m}$ if the numbers of tokens in its input and inhibitor places verify the usual enabling conditions for PN models without priority $(\boldsymbol{m} \geq I(t)) \bigwedge (\boldsymbol{m} < H(t))$;
- a transition t_j is said to *be enabled* in marking $\boldsymbol{m}$ if it has concession in the same marking, and if no transition $t_k \in T$ of priority $\pi_k > \pi_j$ exists that has concession in $\boldsymbol{m}$. As a consequence, two transitions may be simultaneously enabled in a given marking only if they have the same priority level;
- a transition t_j can *fire* only if it is enabled. The effect of transition firing is identical to the case of PN models without priority.

Note that the presence of priority only restricts the set of enabled transitions (and therefore the possibilities of firing) with respect to the same PN model without priority. This implies that some properties are not influenced by the addition of a priority structure, while others are changed in a well-determined manner, as we shall see in a while.

3.1 Conflicts, Confusion and Priority

The notions of conflict and confusion are modified when a priority structure is associated with transitions. It is thus very important to be able to clearly identify by inspection of the net structure the sets of potentially conflicting transitions.

Conflict — The notion of conflict is drastically influenced by the introduction of a priority structure in PN models. The definition of conflict has to be modified with respect to the new notion of concession. Instead, the definition of enabling degree given in Section 2.2 remains unchanged for PN models with priority. Observe that this implies that both, transitions that have concession and enabled transitions, have enabling degree greater than zero. Conflict resolution causes the enabling degree of some transition to be reduced, and this may happen both for transitions with concession and for enabled ones.

[7] Without loss of generality, we also assume that all lower priority levels are not empty:

$$\forall t_j \in T, \quad \pi_j > 0 \implies \exists t_k \in T \ : \ \pi_k = \pi_j - 1$$

The definition of transitions with different priority levels introduces a further complication, since it destroys the locality of conflicts typical of PN models without priority. This observation leads to the possibility of *indirect conflicts*.

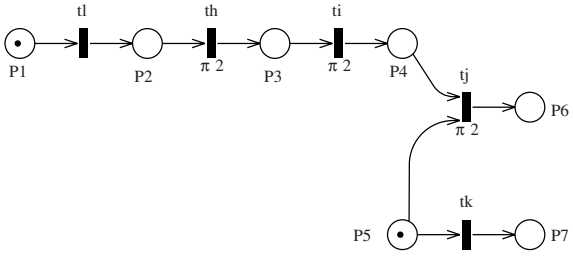


Fig. 6. An example of indirect conflict

Let us consider the net in Fig. 6. Transitions t_l and t_k are both enabled in the marking represented in the figure (since they both have concession, and there is no higher priority transition which has concession), and apparently they are not in conflict, since they do not share input or inhibition places. According to the definition of concurrent transitions given in Section 2.2, one might conclude that t_l and t_k are concurrent. However, the firing of t_l enables a sequence of transitions t_h, t_i, and t_j which have higher priority than t_k, so that:

1. transition t_k becomes disabled while keeping its concession;
2. transition t_h is certainly the next transition to fire;
3. the firing of t_j removes the token from place p_5, thus taking concession away from transition t_k.

This sequence of events is not interruptible after the firing of t_l, due to the priority structure, and eventually results in the disabling of t_k through the firing of higher priority transitions. We call this situation *indirect conflict* between t_l and t_k.

The presence of indirect conflicts complicates the behaviour of PN with priorities and makes their analysis more difficult. The local effect of choices, typical of PN models without priorities, is now replaced by a global feature that could require expensive and estensive testing to identify the markings that can be reached when several transitions have simultaneous concessions. In order to confine this problem, it is possible to define an equivalence relation [6,13] that partitions the set T into equivalence classes called *extended conflict sets*(ECS).

In any marking that enables transitions of the same ECS, a choice that may have effect on the future evolution of the net must be made in order to decide which, among these transitions, has to be fired next. Instead, two simultaneously enabled transitions t_i and t_j that belong to different ECS can be fired in any order and are thus concurrent.

Confusion and priority — We introduced the concept of confusion in the framework of PN models without priority. In this section we shall see how the introduction of a priority structure can avoid confusion.

Confusion is an important notion because it highlights the fact that, in terms of event ordering, the system behaviour is not completely defined: this underspecification could be due either to a precise modelling choice or to a modelling error. The introduction of a priority structure may force a deterministic ordering of conflict resolutions that removes confusion.

For instance, let us consider again the example depicted in Fig. 6 assuming first that transitions t_l and t_k have the same priority level of the others. A confusion situation arises due to the fact that both sequences $\sigma = t_k, t_l, t_h, t_j$ and $\sigma' = t_l, t_h, t_i, t_k$ are fireable in a marking with tokens in places p_1 and p_5, and that they involve different conflict resolutions. By making the priority level of transitions t_h, t_i, and t_j higher than that of t_l and t_k we have removed the confusion situation since any conflict between t_j and t_k is always solved in favour of t_j; as a consequence the sequence $\sigma' = t_l, t_h, t_i, t_k$ is no more fireable in $\boldsymbol{m}$ and confusion is avoided.

Again the interested reader is referred to [6,13] for a deeper discussion of these aspects and for their complete formalizations.

3.2 Properties of Petri Nets with Priority

In order to briefly discuss the impact that priorities have on the properties of PN models, we must first divide them into two broad classes. Properties that hold for all states in the state space are called *safety* or *invariant* properties; properties that instead hold only for some state are called *eventuality* or *progress* properties). Examples of invariant properties are boundedness, and mutual exclusion. Examples of eventuality properties are reachability (a given marking will be eventually reached) and liveness (a transition will eventually become enabled).

Let $\mathcal{M}_\pi$ be a PN with priority and let $\mathcal{M}$ be the underlying PN without priority. Since the introduction of priority can only reduce the state space, all the safety properties that can be shown to be true for $\mathcal{M}$, surely hold also for $\mathcal{M}_\pi$. Eventuality properties instead are not preserved in general by the introduction of a priority structure.

It is interesting to observe that P and T-invariants describe properties that continue to hold after the addition of a priority structure. The reason is that they are computed only by taking into account the state change caused by transition firing, without any assumption on the possibility for a transition of ever becoming enabled. Boundedness is preserved by the introduction of a priority structure in the sense that a bounded PN model remains bounded after the introduction of a priority specification. This implies that the use of P-semiflows to study the boundedness of a PN model can be applied to the model without priority $\mathcal{M}$ associated with a priority PN model $\mathcal{M}_\pi$ and if the former model is shown to be structurally bounded, the conclusion can be extended to the latter one. Observe, however, that an unbounded PN model may become bounded after the specification of an appropriate priority structure.

On the other hand, since enabling is more restricted than in the corresponding PN model without priority, reachability is not preserved in general by the addition of a priority structure. However, a marking $\boldsymbol{m}'$ is reachable from a marking $\boldsymbol{m}$ in a PN model with priority only if it is reachable in the corresponding PN model without priority.

Liveness is intimately related to the enabling and firing rules, hence it is greatly influenced by a change in the priority specification: a live PN model may become not live after the introduction of an inappropriate priority structure and, viceversa, a PN model that is not live, may become live after the addition of a proper priority structure.

4 Time in Petri Nets

In this Section we discuss the issues related to the introduction of *temporal concepts* into PN models. Particular attention will be given to the temporal semantics that is peculiar to stochastic PNs (SPNs) and generalized SPNs (GSPNs).

4.1 The Motivations for Timing

The PN models that were considered in the previous sections included no notion of time. The concept of time was intentionally avoided in the original work by C.A.Petri [47], because of the effect that timing may have on the behaviour of PNs. In fact, the association of timing constraints with the activities represented in PN models may prevent certain transitions from firing, thus destroying the assumption that all possible behaviours of a real system are represented by the structure of the PN.

Soon after their proposal, PNs were recognized as a convenient formalism for the construction of models of real concurrent systems, where the concept of time becomes of paramount importance when the interest is driven by real applications whose efficiency is always a relevant design problem. Indeed, in areas like hardware and computer architecture design, communication protocols, and software system analysis, timing is crucial even to define the logical aspects of the dynamic operations.

Time is thus introduced in PNs to model the interaction among several activities considering their starting and completion instants. The introduction of time specifications corresponds to an interpretation of the model by means of the observation of the autonomous (untimed) model and the definition of a non-autonomous model.

Different ways of incorporating timing information into PN models have been proposed by many researchers.

The pioneering works in the area of timed PNs were performed by J.D.Noe and G.J.Nutt [45], and by P.M.Merlin and D.J.Farber [38].

The different proposals are strongly influenced by the specific application fields and it is possible to find time associated with tokens, places, arcs, and transitions.

4.2 Timed Transitions

Ideally, the introduction of time into PN models, should not modify the behaviour of the underlying untimed model in order to make possible the analysis of the timed PN exploiting the properties of the basic model as well as the available theoretical results. The addition of temporal specifications therefore should not modify the unique and original way of expressing synchronization and parallelism that is peculiar of PNs. This requirement often conflicts with the user's wishes for extensions of the basic PN formalism to allow a direct and easy representation of specific phenomena of interest. A compromise must thus be reached to accomodate these conflict requirements in the best possible way. Time specifications are also used for reducing the non-determinism of the model by means of rules based on time considerations. Finally, time extensions must provide methods for the computation of performance indices.

Timed transitions represent the most common extension used by the authors to add time to PN models. The firing of a transition in a PN model corresponds to the event that changes the state of the real system. This change of state can be due to one of two reasons: it may either results from the verification of some logical condition in the system, or be induced by the completion of some activity. Considering the second case, we note that transitions can be used to model activities, so that transition enabling periods correspond to activity executions and transition firings correspond to activity completions. Hence, time can be naturally associated with transitions.

Different firing policies may be assumed: the *three-phase firing* assumes that tokens are consumed from input places when the transition is enabled, then the delay elapses, finally tokens are generated in output places; *atomic firing* assumes instead that tokens remain in input places during the whole transition delay; they are consumed from input places and generated in output places only when the transition fires.

Timed transition Petri nets (TTPN) with atomic firing can preserve the behaviour of the underlying untimed model. It is thus possible to qualitatively study TTPN with atomic firing exploiting the theory developed for untimed (autonomous) PN (reachability set, invariants, etc.). In TTPN,timing specifications may affect the qualitative behaviour of the PN only when they describe *constant* and *interval* firing delays.

We can explain the behaviour of a timed transition (whose graphical representation is usually a box or a thick bar and whose name usually starts with T) by assuming that it incorporates a timer. When the transition is enabled, its local clock is set to an initial value. The timer is then decremented at constant speed, and the transition fires when the timer reaches the value zero. The timer associated with the transition can thus be used to model the duration of an activity whose completion induces the state change that is represented by the change of marking produced by the firing of T. The type of activity associated with the transition, whose duration is measured by the timer, depends on the system that we are modelling: it may correspond to the execution of a task by a processor, or to the transmission of a message in a communication network, or to the work

performed on a part by a machine tool in a manufacturing system. It is important to note that the activity is assumed to be in progress while the transition is enabled. This means that in the evolution of more complex nets, an interruption of the activity may take place if the transition loses its enabling condition before it can actually fire. The activity may be resumed later on, during the evolution of the net in the case of a new enabling of the associated transition. This may happen several times until the timer goes down to zero and the transition finally fires.

It is possible to define a *timed transition sequence* or *timed execution* of a timed PN system as a transition sequence (as defined in Section 2.2) augmented with a set of nondecreasing real values describing the epochs of firing of each transition. Such a timed transition sequence is denoted as follows:

$$[(\tau_{(1)}, T_{(1)}); \cdots; (\tau_{(j)}, T_{(j)}); \cdots] \tag{10}$$

The time intervals $[\tau_{(i)}, \tau_{(i+1)})$ between consecutive epochs represent the periods during which the PN sojourns in marking $\boldsymbol{m}_{(i)}$ which is then denoted as $\sigma_{(i)} = \tau_{(i+1)} - \tau_{(i)}$. This sojourn time corresponds to a period in which the execution of one or more activities is in progress and the state of the system does not change.

Focussing on markings and sojourn times, the timed transition sequence yields also a *timed marking sequence*

$$[(\sigma_{(1)}, \boldsymbol{m}_{(1)}); \cdots; (\sigma_{(j)}, \boldsymbol{m}_{(j)}); \cdots] \tag{11}$$

where $\sigma_{(1)}$ represents the time spent in marking $\boldsymbol{m}_{(1)}$ that was reached after the firing of transition $T_{(1)}$ and was left upon firing of transition $T_{(2)}$.

4.3 Immediate Transitions

As we noted before, not all the events that occur in a system correspond to the end of time-consuming activities (or to activities that are considered time-consuming at the level of detail at which the model is developed). For instance, in the Producer/Consumer model of Fig. 1, the acquisition of a buffer position, when it is available, requires a very small amount of time compared with the execution of the task of producing the item. The same can be true when we compare context switchings and task durations in the model of a multiprocessor system described at a high level of abstraction, or bus arbitration compared with bus use. In other cases, the state change induced by a specific event may be quite complex, and thus difficult to obtain with the firing of a single transition. Moreover, the state change can depend on the present state in a complex manner. As a result, the correct evolution of the timed PN model can often be described with subnets of transitions that consume no time and encode the logics or the algorithm of state evolution induced by the complex event.

To cope with both these situations, it is convenient to introduce in timed PN, a second type of transitions called *immediate*. Immediate transitions fire as soon as they become enabled (with a null delay), thus acquiring a sort of precedence

over timed transitions. With this assumption we are introducing in the model a priority structure that will be explicitely addressed in the following sections.

In this paper, immediate transitions are depicted as thin bars whereas timed transitions are depicted as boxes or thick bars. Following these considerations, the Producer/Consumer model of Fig. 1 can now be depicted as in Fig. 7 where the immediate transitions are used to represent both the delivery/ withdrawal of items to/from the buffer.

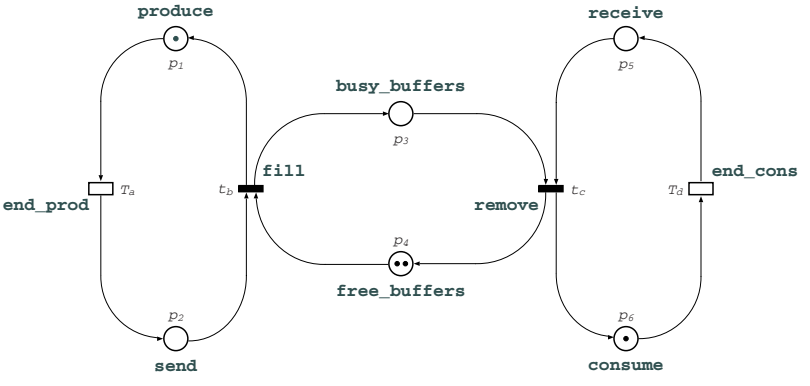


Fig. 7. The Producer/Consumer model with immediate transitions

4.4 Parallelism and Conflict

The introduction of temporal specifications in PN models must not reduce the modelling capabilities with respect to the untimed case. Let us verify this condition as far as parallelism and conflict resolution are considered.

Pure parallelism can be modelled by two transitions that are independently enabled in the same marking. The evolution of the two activities is measured by the decrement of the clocks associated with the two transitions. When one of the timers reaches zero, the transition fires and a new marking is produced. In the new marking, the other transition is still enabled and its timer can either be reset or not depending on the different ways of managing this timer that will be discussed in the next Section.

Consider now transitions T_1 and T_2 in Fig. 8. In this case, the two transitions are in free-choice conflict. In untimed PN systems, the choice of which of the two transitions to fire is completely nondeterministic. When more than one timed transition with atomic firing is enabled, the behaviour is similar, but for the choice two alternative selection rules are possible:

- **preselection** - the enabled transition that will fire is chosen when the marking is entered, according to some metric (e.g., priority),
- **race**- the enabled transition that will fire is the one whose firing delay is minimum.

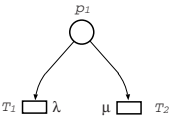


Fig. 8. A simple example of a free-choice conflict among timed transitions

In TTPNs the race policy it usually adopted and this is the alternative that will be used in the rest of this paper when dealing with timed transitions.

Considering now that the presence of immediate transitions introduces a priority structure, we must observe that timed transitions are assumed to be all associated with the lowest possible level of priority. Immediate transitions can instead be characterized by different (higher) priority levels and the concept of concession discussed in Section 3 must be used when resolving conflicts.

When a conflict involves transitions of different priority levels, the choice is deterministically made in favour of the higher priority ones, which are also those who are enabled. When several immediate transitions are enabled in the same marking the choice of the transition that will fire is made using a mechanisms that consists in the association of a discrete probability distribution function with the set of conflicting transitions, so that the conflict among immediate transitions is randomly solved.

The priority that immediate transitions have over timed one can be used to separate conflict resolution from timing specification of transitions. The conflict can be transferred to a barrier of conflicting immediate transitions, followed by a set of timed transitions. The extensive use of this technique can eliminate from a net all conflicts among timed transitions that are simultaneously enabled in a given marking. If this mechanism is consistently used to prevent timed transitions from entering into conflict situations, a *preselection* policy of the (timed) transition to fire next is said to be used.

Conflicts comprising timed and immediate transitions have an important use in timed PNs, since they allow the interruption (or preemption) of ongoing activities, when some special situation occurs. Consider, for example, the subnet in Fig. 9. A token in place p_1 starts the activity modelled by timed transition T_1. If a token arrives in p_2 before the firing of T_1, immediate transition t_2 becomes enabled and (due to its higher priority) fires, thus disabling timed transition T_1.

The presence of immediate transitions induces a distinction among markings. Markings in which no immediate transitions are enabled are called *tangible*,

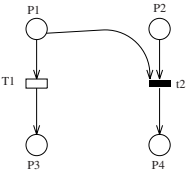


Fig. 9. Interrupting an activity with an immediate transition

whereas markings enabling at least one immediate transition are said to be *vanishing*. From this observation follows that TTPNs spend a positive amount of time in tangible markings, and a null time in vanishing markings.

4.5 Memory

An important issue that arises at every transition firing, when timed transitions are used in a model, is how to manage the timers of all the transitions that do not fire.

From the modeling point of view, the different policies that can be adopted to manage these situations link the past history of the system to its future evolution considering various ways of retaining *memory* of the time already spent in performing the activities associated with all the transitions. The question concerns the *memory policy* of transitions, and defines how to set the transition timers when a state change occurs, possibly modifying the enabling of transitions. Two basic mechanisms can be considered for a timed transition at each state change.

- **Continue.** The timer associated with the transition holds the present value and will *continue* later on its count-down.
- **Restart.** The timer associated with the transition is *restarted*, i.e., its present value is discarded and a new value will be generated when needed.

To model the different behaviours arising in real systems, different ways of keeping track of the past are possible by associating different continue or restart mechanisms with timed transitions. We discuss here three alternatives:

- **Resampling.** At each and every transition firing, the timers of all the timed transitions are discarded (restart mechanism). No memory of the past is recorded. After discarding all the timers, new values of the timers are set for the transitions that are enabled in the new marking.
- **Enabling memory.** At each transition firing, the timers of all the timed transitions that are disabled are discarted (resampling mechanism) whereas the timers of all the timed transitions that are not disabled hold their present value (continue mechanism). The memory of the past is recorded with an *enabling memory variable* associated with each transition. The enabling memory variable accounts for the work performed by the activity associated with the transition since the last instant of time its timer was set. In other words, the enabling memory variable measures the enabling time of the transition since the last instant of time it became enabled.
- **Age memory.** At each transition firing, the timers of all the timed transitions hold their present values (continue mechanism). The memory of the past is recorded with an *age memory variable* associated with each timed transition. The age memory variable accounts for the work performed by the activity associated with the transition since the time of its last firing. In other words, the age memory variable measures the *cumulative* enabling time of the transition since the last instant of time when it fired.

These three memory policies can be used in timed PN models for different modelling purposes. In the first case (resampling) the work performed by activities associated with transitions that do not fire is lost. This may be adequate for modelling, for example, competing activities of the type one may find in the case of the parallel execution of hypothesis tests. The process that terminates first is the one that verified the test; those hypotheses whose verification was not completed become useless, and the corresponding computations need not be saved. The practical and explicit use of this policy is very limited, but it must be considered because of its theoretical importance in the case of SPNs and GSPNs.

The other two policies are of greater importance from the application viewpoint. They can coexist within the same TTPN model, because of the different semantics that can be assigned to the different transitions of the model. For a detailed discussion on this topic the reader is referred to [2,26].

4.6 Multiple Enabling

Special attention must be paid to the timing semantics in the case of timed transitions with enabling degree larger than one. Different semantics are possible when several tokens are present in the input places of a transition. Borrowing from queueing network terminology, we can consider the following different situations:

1. **Single-server semantics** - a firing delay is set when the transition is first enabled, and new delays are generated upon transition firing if the transition is still enabled in the new marking.
2. **Infinite-server semantics** - every enabling set of tokens is processed as soon as it forms in the input places of the (timed) transition. Its corresponding firing delay is generated at this time, and the timers associated with all these enabling sets run down to zero in parallel.
3. **Multiple-server semantics** - enabling sets of tokens are processed as soon as they form in the input places of the transition up to a maximum degree of parallelism (say K). For larger values of the enabling degree, the timers associated with new enabling sets of tokens are set only when the number of concurrently running timers decreases below the value of K.

The introduction of these different firing semantics permits the definition of PN models that are graphically simple without losing any of the characteristics that allow the analysis of their underlying behaviours.

5 Stochastic Petri Nets

Timed Petri nets in which the firing delays are specified by random variables yeld to probabilistic models. The execution of a timed PN model of this type corresponds to a realization of a stochastic point process.

The use of negative-exponential distributions for the definition of temporal specifications is particularly attractive because TTPNs in which all the transition delays are exponentially distributed can be mapped onto continuous-time

Markov chains (CTMC). In this case the memoryless property of the negative-exponential distribution makes unnecessary the distinction between the distribution of the delay itself, and the distribution of the remaining delay after a change of state, as we shall see in a while.

Stochastic Petri Nets (SPNs) are TTPNs with atomic firing and in which transition firing delays are negative-exponentially distributed random variables: each transition T_i is associated with a random firing delay whose probability density function is a negative-exponential with rate w_i.

SPNs were originally defined in [31,40] Formally, a SPN model is a 6-tuple

$$\text{SPN} = (P, T, I(.), O(.), W(.), \boldsymbol{m}_0) \tag{12}$$

P, T, $I(.)$, $O(.)$, and $\boldsymbol{m}_0$ have the usual meanings so that the underlying PN model constitutes the structural component of a SPN model.

The function W allows the definition of the stochastic component of a SPN model mapping transitions into real positive functions of the SPN marking. Thus, for any transition T it is necessary to specify a function $W(T, \boldsymbol{m})$. In the case of marking independency, the simpler notation w_k is normally used to indicate $W(T_k)$, for any transition $T_k \in T$. The quantity $W(T_k, \boldsymbol{m})$ (or w_k) is called the "rate" of transition T_k in marking $\boldsymbol{m}$.

In this section we show how SPNs can be converted into Markov chains and how their analysis can be performed to compute interesting performance indices. The construction of the Markov chain associated with a SPN is described first, to set the ground for the subsequent derivation of the probabilistic model associated with a GSPN. Only SPNs with finite state space are considered, as they yield Markov chains that are solvable with standard numerical techniques. More advanced solution methods are discussed in [32,44,14].

5.1 The Stochastic Process Associated with a SPN

We have already discussed the motivations behind the work of several authors that led to the proposal of Timed and Stochastic Petri Nets. Due to the memoryless property of the negative-exponential distribution of firing delays, it is relatively easy to show [31,40] that SPNs are isomorphic to CTMCs. In particular, a k-bounded SPN system can be shown to be isomorphic to a finite CTMC.

Finite State Machine and Marked Graph SPNs - This can be easily seen when the structure of the SPN is that of both a *finite state machine* (no transition has more than one input and one output place) and of a *marked graph* (no place has more than one input and one output transition) with only one token in its initial marking (see Fig. 10). In this case each place of the net univocally corresponds to a state of the model and the position of the token at a given instant of time identifies the state of the model at that same time. Each place of the net maps into a state of the corresponding Continuous Time Stochastic Process and each transition maps into an arc annotated by the rate

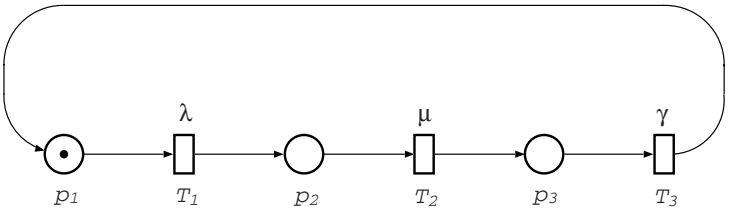


Fig. 10. A simple Stochastic Petri net of the MG type

of the corresponding firing time distribution. Moreover, if the firing times of the transitions have negative-exponential distributions and if the structure of the net is that of a marked graph, the time spent in each place by the net is completely identified by the characteristics of the only transition that may withdraw the token from that place. The sojourn time in that place has a negative exponential distribution and thus the Stochastic Process associated with the net is a CTMC (see Fig. 11). When the net has the structure of a finite state machine (e.g, the model of Fig. 12), conflicts among simultaneously enabled transitions arise since several transitions may share the same input place. Since we are assuming that all the activities have negative-exponentially distributed durations, the CTMC corresponding to the SPN is obtained from the net in a straightforward manner. Again each place of the SPN maps into a state of the corresponding CTMC and each transition of the SPN maps into an arc of the CTMC annotated with the rate of the corresponding firing time distribution. Also in this case the time spent by the net in each place has a negative-exponential distribution, but its rate is given by the sum of the firing rates of all the transitions that withdraw tokens from that place (see Fig. 13).

More complex situations arise, even in these simple cases, when several tokens are allowed in the initial marking. Consider the case of the net in Fig. 14 and its RG represented in Fig. 15. Assume that all the transitions of the net operate according to a "single-server" semantics and that, upon firing of a multiply enabled transition, tokens are withdrawn from its input places selecting them at random (see Section 4.6). Starting from the initial marking $(2, 0, 0, 0)$, we get to marking $(1, 1, 0, 0)$ when the only enabled transition (T_1) fires. In that precise moment, the two system activities represented by transitions T_1 and T_2

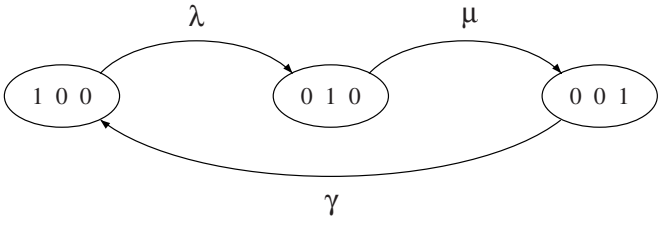


Fig. 11. Reachability Graph and Markov Chain of the net of fig. 10

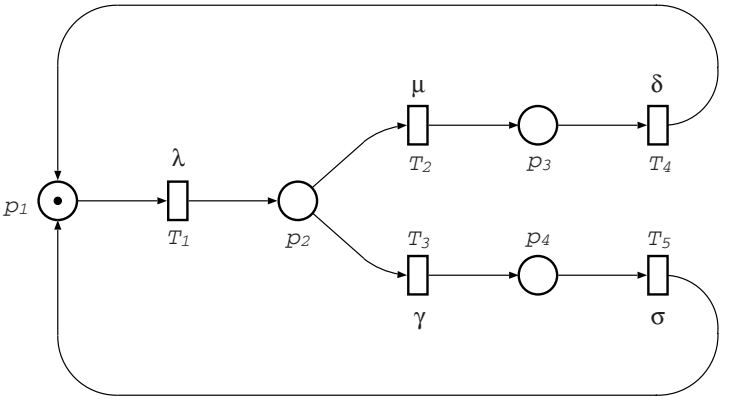


Fig. 12. A simple Stochastic Petri net of the FSM type

start and their durations are chosen. Since these two durations are samples of continuous random variables whose distributions do not have discontinuities, the probability of chosing two identical values is null and thus the net leaves marking $(1, 1, 0, 0)$ when the fastest of the two transitions fires. Suppose that this "race" is won by transition T_2. When the net enters marking $(1, 0, 1, 0)$, an interesting situation occurs. We have again two possible ways out from this last marking, corresponding to the (possible) firings of transitions T_1 and T_3 (as illustrated by the RG of Fig. 15). Looking at this situation more in detail, we observe that the "race" is now between the completion time of the activity associated with transition T_1 - that started already when the net was in the previous marking $(1, 1, 0, 0)$ - and that of the activity corresponding to T_3 that started just at the entering of the net in this last marking. Because of the "memoryless" property of the negative-exponential distributions of the firing times of all the transitions involved in this last race (as well as in all the other races that can be envisioned during the evolution of the net), the distribution of the remaining firing time

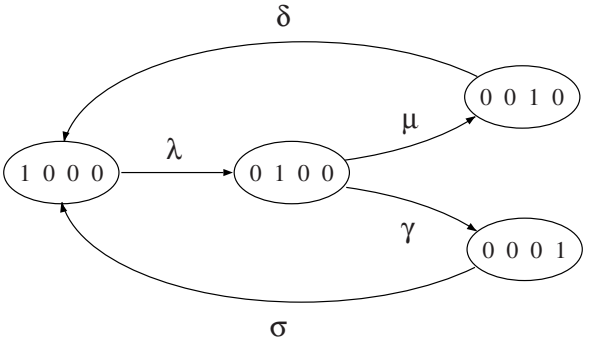


Fig. 13. Reachability Graph and Markov Chain of the net of fig. 12

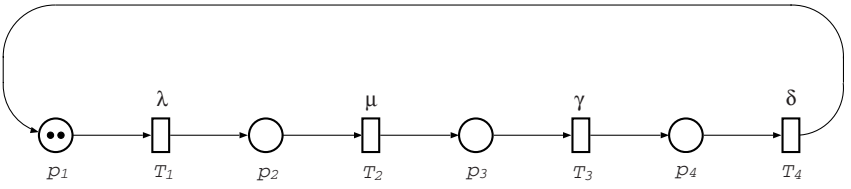


Fig. 14. A simple Stochastic Petri net with multiple tokens of the

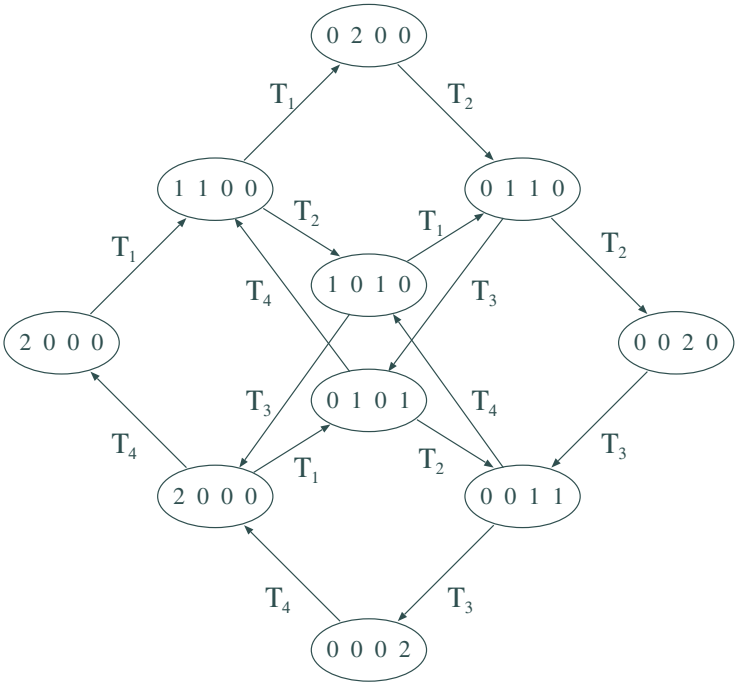


Fig. 15. Reachability Graph of the net of fig. 14

of transition T_1 is identical to that of its total firing time and the probability that transition T_1 has of winning the race is determined on the basis of the rates of its (total) firing time distribution as well as of that of transition T_3. This is the argument that allows to identify the stochastic process that underlies the behaviour of this net as a Markov chain. Also in this case the state transition diagram is still isomorphic to the RG of the PN with the state transition arcs annotated with the firing rates of the corresponding transitions as indicated in Fig. 16. Notice that transitions T_1 and T_3 are concurrent and do not interfere in their firing processes. This is perfectly captured by the RG of the net where, from marking $(1, 0, 1, 0)$ it is possible to reach marking $(0, 1, 0, 1)$ firing the two transitions T_1 and T_3 in different orders (via intermediate markings $(0, 1, 1, 0)$ or

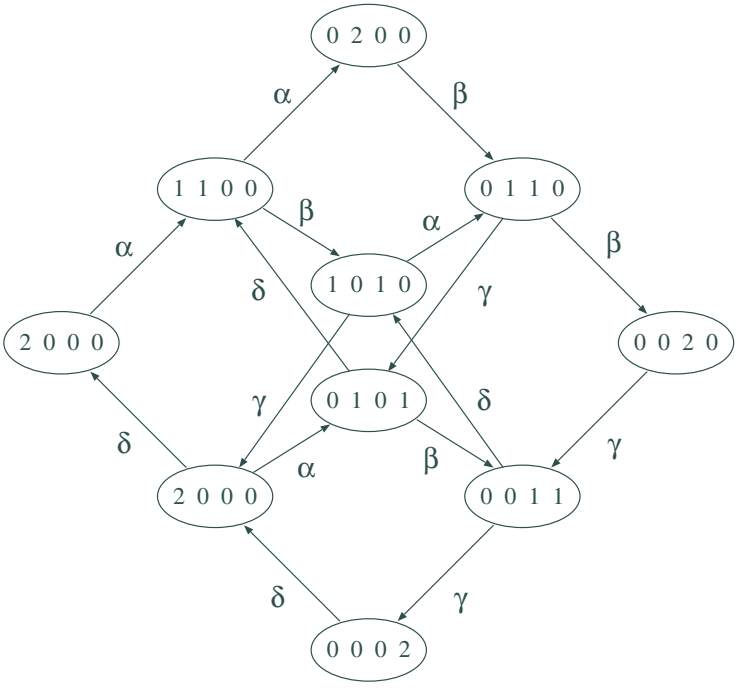


Fig. 16. Markov Chain deriving from the net of fig. 14

$(1, 0, 0, 1)$, respectively). Instead, by labeling with the corresponding transition rates the arcs of the state transition diagram of Fig. 16, we implicitly say that, every time a state changes, the activities that turn out to be active in the new state are started anew.

Formally, the Markov property applied to the stochastic process $\{X(t), t \geq 0\}$ representing the execution of the SPN says that

$$Pr\{X_{n+1} = \boldsymbol{m}_j, \sigma_j > t | (X_n = \boldsymbol{m}_i, \sigma_i = u), (X_{n-1} = \boldsymbol{m}_k, \sigma_k = v), ...\} = Pr\{X_{n+1} = \boldsymbol{m}_j, \sigma_j > t | X_n = \boldsymbol{m}_i\} \quad (13)$$

where $X_n = X(\tau_{(n)}+)$, $n \geq 0$ is the marking reached by the net immediately after the firing of the $n-th$ transition and σ_j is the sojourn time in marking mb_j as defined for the timed marking sequence introduced in Section 4.2.

The interpretation of this expression in terms of the behaviours of the transitions enabled in a given marking and whose firings are the actual cause of the timed execution of the net, requires some additional notation and definitions. Recalling from Section 2.2 that $E(\boldsymbol{m})$ represents the set of transitions enabled in marking $\boldsymbol{m}$, we can observe that $E(\boldsymbol{m}) = NE(\boldsymbol{m}) \cup OE(\boldsymbol{m})$, where $NE(\boldsymbol{m})$ represents the set of transitions that became enabled when the net entered marking $\boldsymbol{m}$ (and were not enabled in the previous marking), and $OE(\boldsymbol{m})$ is instead the set of transitions that were enabled in the previous marking and are still

enabled in marking $\boldsymbol{m}$. Denoting with $\delta_k, k \in E(\boldsymbol{m}_j)$ the firing times (or firing delays) of all the transitions enabled in marking $\boldsymbol{m}_j$, we can observe that

$$Pr\{\sigma_j > t\} \;=\; Pr\left\{ \bigcap_{k \in E(\mathbf{m}_j)} \delta_k > t \right\} \tag{14}$$

so that,

$$\begin{aligned} & Pr\{X_{n+1} = \boldsymbol{m}_j, \sigma_j > t | X_n = \boldsymbol{m}_i, \sigma_i > u\} \;= \\ &= \frac{Pr\{X_{n+1} = \boldsymbol{m}_j, \sigma_j > t, \sigma_i > u | X_n = \boldsymbol{m}_i\}}{\Pr\{\sigma_i > u\}} \\ &= \frac{Pr\left\{X_{n+1} = \boldsymbol{m}_j, \left(\bigcap_{k \in NE(\mathbf{m}_j)} \delta_k > t \right), \left(\bigcap_{h \in OE(\mathbf{m}_j)} \delta_h > (t+u) \right) | X_n = \boldsymbol{m}_i \right\}}{Pr\left\{ \bigcap_{h \in OE(\mathbf{m}_j)} \delta_h > u \right\}} \\ &= Pr\left\{X_{n+1} = \boldsymbol{m}_j, \left(\bigcap_{k \in E(\mathbf{m}_j)} \delta_k > t \right) | X_n = \boldsymbol{m}_i \right\} \end{aligned} \tag{15}$$

where the last step is made possible by the memory-less property of the negative exponential distribution. From this result, we can conclude that indeed in these cases, whenever an activity started in a previous marking and continues through several (intermediate) markings up to the current one, its duration before entering the current marking is irrelevant for any probabilistic evaluation of the future execution of the net.

This result is the basis for the observation that in SPN with negative-exponential distributions of the transition's firing times, the model behaves as if the corresponding system followed a *global resampling policy* so that all the activities start anew every time the system enters a state that enables them (even if they were already enabled in the previous one).

The above discussion of the execution of the SPN of Fig. 14 purposely disregarded the situation in which several tokens were in the input place of a single transition. However, even the very simple case of two tokens waiting in the only input place of a transition raises another set of interesting questions that must be addressed in developing the corresponding probabilistic model. The first has to do with the speed at which the transition withdraws the tokens from its input place in this situation and thus with the service policies discussed in Section 4.6. In the most general case of assuming a form of *load dependency* in which the firing rate of the transition is a function of its enabling degree, an additional specification must be introduced in the model to define the load dependency function associated with each transition. A second question refers to the selection of the token that is removed from the input place upon the firing of the transition. From a "classical" PN point of view, this selection policy is inessential since tokens do not carry any identity. In many applications however, it is convenient to associate a physical meaning with the tokens (e.g., customers), so that questions on their flow through the net can be answered. In these situations, when several tokens are simultaneously present in the input place of a transition, if this is assumed to operate with a single server policy, a question on

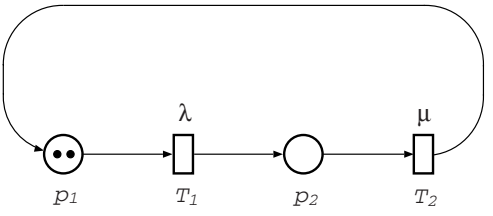


Fig. 17. A simple SPN with two timed transitions

the queueing policy applied to these tokens becomes interesting. As we already notice before, the most natural policy is a *random order*. When the firing times are exponentially distributed and when the performance figures of interest are only related to the moments of the number of tokens in the input place of a transition, it is possible to show that many queueing policies yield the same results (e.g., random, FIFO, LCFS). It must however be observed that in other cases the choice of the queueing policy may be important and that policies different from the random one must be explicitly implemented through appropriate net constructions.

Considering as an example the net of Fig. 17, we can construct the CTMCs of Fig. 18 (a) and (b) dependi9ng on whether we assume that the transitions perform with single-server or multiple-server semantics

SPNs with general structure - Drawing from the discussion done so far, it is possible to conclude that the CTMC associated with a given SPN system is obtained by applying the following simple rules:

1. The CTMC state space $S = \{s_i\}$ corresponds to the reachability set $RS(\boldsymbol{m}_0)$ of the PN associated with the SPN ($\boldsymbol{m}_i \leftrightarrow s_i$).
2. The transition rate from state s_i (corresponding to marking $\boldsymbol{m}_i$) to state s_j ($\boldsymbol{m}_j$) is obtained as the sum of the firing rates of the transitions that are enabled in $\boldsymbol{m}_i$ and whose firings generate marking $\boldsymbol{m}_j$.

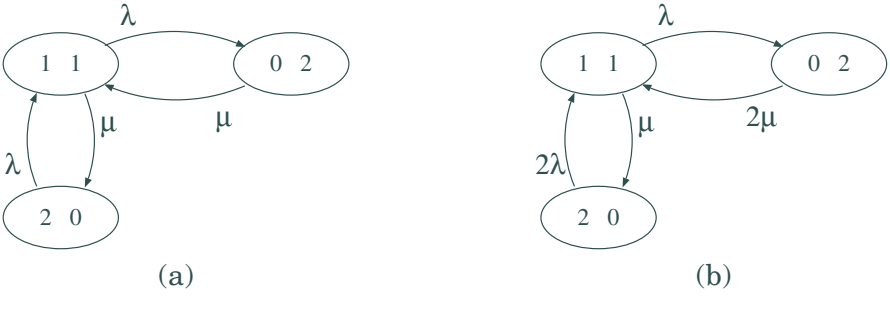


Fig. 18. Markov Chain deriving from the net of fig. 17

Based on these simple rules, it is possible to devise algorithms for the automatic construction of the infinitesimal generator (also called the state transition rate matrix) of the isomorphic CTMC, starting from the SPN description.

Assuming that all the transitions of the net operate with a single-server semantics and marking-independent speeds, and denoting with $\mathbf{Q}$ this matrix, with w_k the firing rate of T_k, and with $E_j(\boldsymbol{m}_i) = \{h : T_h \in E(\boldsymbol{m}_i) \wedge \boldsymbol{m}_i[T_h\rangle \boldsymbol{m}_j\}$ the set of transitions whose firings bring the net from marking $\boldsymbol{m}_i$ to marking $\boldsymbol{m}_j$, the components of the infinitesimal generator are:

$$q_{ij} = \begin{cases} \sum_{T_k \in E_j(\boldsymbol{m}_i)} w_k & i \neq j \\ -q_i & i = j \end{cases} \tag{16}$$

where

$$q_i = \sum_{T_k \in E(\boldsymbol{m}_i)} w_k \tag{17}$$

Let $\pi(\boldsymbol{m}_i, \tau)$ be the probability that the SPN is in marking $\boldsymbol{m}_i$ at time τ. The Chapman-Kolmogorov equations for the CTMC associated with an SPN are specified by:

$$\frac{d\pi(\boldsymbol{s}_i, \tau)}{d\tau} = \sum_{\boldsymbol{s}_k} \pi(\boldsymbol{s}_k, \tau) q_{kj} \tag{18}$$

In matrix notation this becomes

$$\frac{d\boldsymbol{\pi}(\tau)}{d\tau} = \boldsymbol{\pi}(\tau)\boldsymbol{Q}, \tag{19}$$

whose solution can be formally written as

$$\boldsymbol{\pi}(\tau) = \boldsymbol{\pi}(0) e^{\boldsymbol{Q}\tau} \tag{20}$$

where $\boldsymbol{\pi}(0)$ is the probability of the initial distribution (in our case we usually have $\pi_i(0) = 1$ if $\boldsymbol{m}_i = \boldsymbol{m}_0$ and $\pi_i(0) = 0$ otherwise) and $e^{\boldsymbol{Q}\tau}$ is the matrix exponentiation formally defined by

$$e^{\boldsymbol{Q}\tau} = \sum_{k=0}^{\infty} \frac{(\boldsymbol{Q}\tau)^k}{k!} \tag{21}$$

In this paper we consider only SPNs originating homogeneous and ergodic CTMC. A k-bounded SPN system is said to be ergodic if it generates an ergodic CTMC; it is possible to show that a SPN system is ergodic if $\boldsymbol{m}_0$, the initial marking, is a home state (see Section 2.2).

If the SPN is ergodic, the steady-state probability distribution on its markings exists and is defined as the limit $\boldsymbol{\pi} = \lim_{\tau \to \infty} \boldsymbol{\pi}(\tau)$. Its value can be computed solving the usual system of linear equations:

$$\begin{cases} \boldsymbol{\pi}\, \boldsymbol{Q} = \mathbf{0} \\ \boldsymbol{\pi}\, \mathbf{1}^{\mathrm{T}} = 1 \end{cases} \tag{22}$$

where $\mathbf{0}$ is a vector of the same size as $\boldsymbol{\pi}$ and with all its components equal to zero and $\mathbf{1}^{\mathrm{T}}$ is a vector (again of the same size as $\boldsymbol{\pi}$) with all its components equal to one, used to enforce the normalization condition.

To keep the notation simple, in the rest of the paper we will use $\pi_i(\tau)$ and π_i instead of $\pi(\boldsymbol{m}_i, \tau)$ and $\pi(\boldsymbol{m}_i)$ to denote the transient and steady state probabilities of marking $\boldsymbol{m}_i$. As we already observed, the Markovian property of this model ensures that the sojourn time in the $i-th$ marking is exponentially distributed with rate q_i. The pdf of the sojourn time in a marking corresponds to the pdf of the minimum among the firing times of the transitions enabled in the same marking; it thus follows that the probability that a given transition $T_k \in E(\boldsymbol{m}_i)$ fires (first) in marking $\boldsymbol{m}_i$ can be expressed as follows:

$$P\{T_k|\boldsymbol{m}_i\} \;=\; \frac{w_k}{q_i}. \tag{23}$$

Using the same argument, we can observe that the average sojourn time in marking $\boldsymbol{m}_i$ is given by the following expression:

$$SJ_i \;=\; E[\sigma_i] \;=\; \frac{1}{q_i}. \tag{24}$$

SPN performance indices - The steady-state distribution $\boldsymbol{\pi}$ is the basis for a quantitative evaluation of the behaviour of the SPN that is expressed in terms of performance indices. These results can be computed using a unifying approach in which proper index functions (also called *reward functions*) are defined over the markings of the SPN and an average reward is derived using the steady-state probability distribution of the SPN. Assuming that $r(\boldsymbol{m})$ represents a reward function, the average reward can be computed using the following weighted sum:

$$E[R] \;=\; \sum_{\boldsymbol{m}_i \in RS(\boldsymbol{m}_0)} r(\boldsymbol{m}_i)\ \pi_i \tag{25}$$

Different interpretations of the reward function can be used to compute different performance indices. In particular, assuming $r(\boldsymbol{m}) = n$ iff $m(p_j) = n$, Eq. 25 yields the *expected value of the number of tokens in place* p_j; instead, when $r(\boldsymbol{m}) = w_j$ iff $T_j \in E(\boldsymbol{m})$ Eq. 25 provides the *mean number of firings per unit of time of transition* T_j, i.e. the *throughput* of transition T_j.

For a detailed discussion of the reward method for computing the performance indices of SPN models, the interested reader is referred to [27]

6 Generalized Stochastic Petri Nets

Several reasons suggest the introduction of the possibility of using immediate transitions into PN models together with timed transitions. As we observed in Section 4.3 the firing of a transition may describe either the completion of a time-consuming activity, or the verification of a logical condition. It is thus natural to use timed transitions in the former case, and immediate transitions in

the latter. Moreover, when all transitions are timed the temporal specification of the model must in some cases consider at one time both the timing and the probability inherent in a choice. It seems natural to provide a way to separate the two aspects in the modelling paradigm, so as to simplify the model specification. Furthermore, by allowing the use of immediate transitions, some important benefits can be obtained in the model solution. They will be described in detail later in this section; we only mention here the fact that the use of immediate transitions may significantly reduce the cardinality of the reachability set, and may eliminate the problems due to the presence of timed transitions with rates that differ by orders of magnitude. The latter situation results in so-called "stiff" stochastic processes, that are quite difficult to handle from a numerical viewpoint. On the other hand, the introduction of immediate transitions in an SPN does not raise any significant complexity in the analysis, as we shall see soon.

SPN models in which immediate transitions coexist with timed transitions with race policy and random firing delays with negative exponential distributions are known by the name of *Generalized SPNs* (GSPNs) [4].

Immediate transitions are fired with priority over timed transitions. Thus, if timing is disregarded, the resulting PN model comprises transitions at different priority levels. The adoption of the race policy may seem to imply the priority of immediate over timed transitions; this is indeed the case in most situations, but the explicit use of priority simplifies the development of the theory, as we shall discuss in a while.

Recall that markings in the reachability set can be classified as *tangible* or *vanishing*. A marking in which no transition is enabled is tangible (but is also an absorbing state - a deadlock - for the execution of the net). The time spent in any vanishing marking is deterministically equal to zero, while the time spent in tangible markings is positive with probability one.

To describe the GSPN dynamics, we separately observe the timed and the immediate behaviour, hence referring to tangible and vanishing markings, respectively. Let us start with the timed dynamics (hence with tangible markings); this is identical to the dynamics in SPNs, that was described before. We can assume that each timed transition has a timer. The timer is set to a value that is sampled from the negative exponential distribution associated with the transition, when the transition becomes enabled for the first time after firing. During all time intervals in which the transition is enabled, the timer is decremented. Transitions fire when their timer reading goes to zero.

With this interpretation, each timed transition can be used to model the execution of some activity in a distributed environment; all enabled activities execute in parallel (unless otherwise specified by the PN structure) until they complete. At completion time, activities induce a change of state, that is limited to their local environment. No special mechanism is necessary for the resolution of timed conflicts: the temporal information provides a metric that allows the conflict resolution.

In the case of vanishing markings, the GSPN dynamics consumes no time: everything takes place instantaneously. This means that if only one immediate transition is enabled, it fires and the following marking is produced. If several immediate transitions are enabled, a metric is necessary to identify which transition will produce the marking modification. Actually, the selection of the transition to be fired is relevant only in those cases in which a conflict must be resolved. If the enabled transitions are concurrent, they can be fired in any order. For this reason, GSPNs associate *weights* with immediate transitions belonging to the same conflict set.

For the time being, let us consider only free-choice conflict sets; the case of non-free-choice conflict sets will be considered later on, but we can anticipate at this point that it can be tackled as the free-choice case by exploiting the definition of ECS introduced in [18,6] and briefly discussed in Section 3.1. The transition weights are used to compute the firing probabilities of the simultaneously enabled transitions comprised within the conflict set. The restriction to free-choice conflict sets guarantees that transitions belonging to different conflict sets cannot disable each other, so that the selection among transitions belonging to different conflict sets is not necessary.

We can thus observe a difference between the specification of the temporal information for timed transitions and the specification of weights for immediate transitions. The temporal information associated with a timed transition depends only on the characteristics of the activity modelled by the transition and thus does not require. information on the other (possibly conflicting) timed transitions, or on their temporal characteristics. On the contrary, for immediate transitions, the specification of weights must be performed considering at one time all transitions belonging to the same conflict set. Indeed, weights are normalized to produce probabilities by considering all enabled transitions within a conflict set. This is not a modelling difficulty when free-choice conflicts among immediate transitions are considered. However, when employing non-free-choice conflicts of immediate transitions, the user has the possibility of describing a much wider range of dynamic behaviours in vanishing markings, but he must be able to correctly associate the immediate transitions with the metrics that define their probabilistic conflict resolution. In principle, this requires the knowledge of the sets of simultaneously enabled non-concurrent immediate transitions in any vanishing marking. This knowledge may not be easy to obtain without the generation of the reachability set, which is however very costly in most cases. The definition of ECSs was introduced to provide the user with the information on the sets of transitions that *may* be in effective conflict (either direct or indirect), thus helping in the definition of weights.

6.1 The Definition of a GSPN Model

GSPNs were originally defined in [4]. The definition was later improved to better exploit the structural properties of the modelling paradigm [3]. The definition we present here is based on the version contained in this second proposal.

Formally, a GSPN model is an 8-tuple

$$\text{GSPN} = (P, T, \Pi(.), I(.), O(.), H(.), W(.), \boldsymbol{m}_0) \tag{26}$$

where $\text{PN}_\pi = (P, T, \Pi(.), I(.), O(.), H(.), \boldsymbol{m}_0)$ is the marked PN with priority underlying the GSPN and $W(.)$ is a function defined on the set of transitions Timed transitions are associated with priority zero, whereas all other priority levels are reserved for immediate transitions.

The underlying PN model constitutes the structural component of a GSPN model, and it must be confusion-free (see Section 3.1) at priority levels greater than zero (i.e., in subnets of immediate transitions).

The function W allows the definition of the stochastic component of a GSPN model as in the case of SPNs. The quantity $W(t_k, \boldsymbol{m})$ (or w_k) is called the "rate" of transition t_k in marking $\boldsymbol{m}$ if t_k is timed, and the "weight" of transition t_k in marking M if t_k is immediate.

Referring again to the Producer/Consumer case of Fig. 7, Table 4 provides the formal specification of the model.

Since in any marking all firing delays of timed transitions have a negative exponential distribution, and all the delays are independent random variables, the sojourn time in a tangible marking is a random variable with a negative exponential distribution whose rate is the sum of the rates of all enabled timed

Table 4. Formal specification of the Producer/Consumer GSPN of Fig. 7

Set of places:	$P = (p_1,\ p_2,\ p_3,\ p_4,\ p_5,\ p_6)$
Set of transitions:	$T = (T_a,\ t_b,\ t_c,\ T_d)$
Priorities:	$\Pi = (\pi_0,\ \pi_1,\ \pi_1,\ \pi_0)$
Weights:	$W = (\alpha,\ 1,\ 1,\ \beta)$
Incidence matrix:	$C =$

$$\begin{array}{c|cccc} & a & b & c & d \\ \hline 1 & -1 & +1 & & \\ 2 & +1 & -1 & & \\ 3 & & +1 & -1 & \\ 4 & & -1 & +1 & \\ 5 & & & -1 & +1 \\ 6 & & & +1 & -1 \end{array}$$

Initial marking: $\boldsymbol{m}_0 = (1,\ 0,\ 0,\ 2,\ 0,\ 1)$

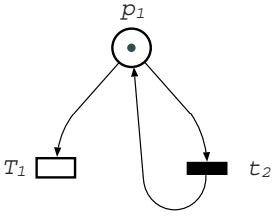


Fig. 19. A conflict set comprising a timed and an immediate transition

transitions in that marking. In the case of vanishing markings, the weights of the immediate transitions enabled in an ECS can be used to determine which immediate transition will actually fire. The choice among immediate transitons belonging to different ECSs is irrelevant because the confusion-free restriction on the immediate subnets makes them truly concurrent. This confusion-free restriction has also a beneficial impact on the model definition, since the association of weights with immediate transitions requires only the information about ECSs, not about reachable markings. For each ECS the analyst thus defines a *local* association of weights from which probabilities are then derived.

As we said before the race policy seems to naturally solve also the conflicts between timed and immediate transitions since immediate transitions will always win the race except for the case of sampling a zero delay from a timed transition with negative exponential distribution that happens with probability zero.

Despites this observation, some problem may arise when the conflict set is enabled infinitely often in a finite time interval. For example, in the case of Fig. 19, the timed and the immediate transitions are always enabled, because the firing of the immediate transition t_2 does not alter the PN marking. The situation is changed only when the timed transition T_1 fires. Even if it will require in general an infinite number of firings of the immediate transition, this happens with probability one in zero time. To avoid these (sometimes strange) limiting behaviours, the priority of immediate over timed transitions was introduced in the GSPN definition.[8]

The RS and the RG of our Producer/Consumer GSPN are represented in Table 5 and Fig. 20, respectively.

6.2 The Stochastic Process Associated with a GSPN

As we have just observed, GSPNs adopt the same firing policy of SPNs; when several transitions are enabled in the same marking, the probabilistic choice of the transition to fire next depends on parameters that are associated with these same transitions and that are not functions of time. The general expression

[8] Notice that this assumption makes the model of Fig. 19 somehow "pathological" from a different point of view since the token remains "trapped" in p_1 and transition T_1 will never fire.

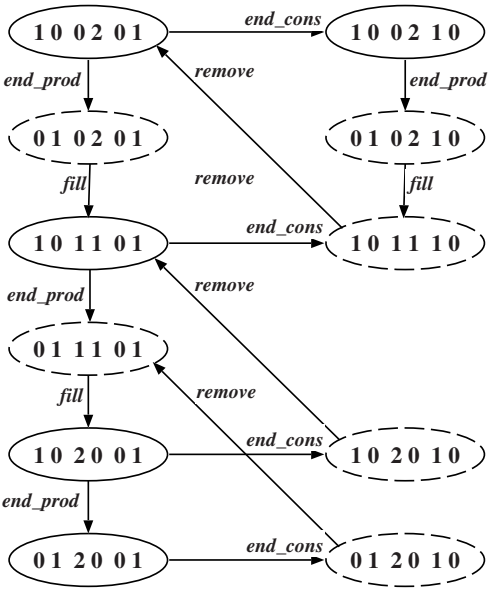


Fig. 20. Reachability Graph of the PN of Fig. 7

for the probability that a given (timed or immediate) transition t_k, enabled in marking $\boldsymbol{m}_i$, fires is:

$$P\{t_k|\boldsymbol{m}_i\} \;=\; \frac{w_k}{q_i} \tag{27}$$

where q_i is the quantity defined by Equation (17). Equation (27) represents the probability that transition t_k fires first, and is identical to Equation (23) for SPNs, with a difference in the meaning of the parameters w_k. When the marking is vanishing, the parameters w_k are the weights of the immediate transitions enabled in that marking and define the selection policy used to make the choice. When the marking is tangible, the parameters w_k of the timed transitions enabled in that marking are the rates of their associated negative exponential distributions. The average sojourn time in vanishing markings is zero, while the average sojourn time in tangible markings is given by Equation (24).

Observing the evolution of a GSPN, we can notice that the distribution of the sojourn time in an arbitrary marking can be expressed as a composition of negative exponential and deterministically zero distributions: we can thus recognize that the marking process $\{\mathcal{M}(\tau), \tau \geq 0\}$ is semi-Markov.

When several immediate transitions are enabled in the same vanishing marking, deciding which transition to fire first makes sense only in the case of conflicts. If these immediate transitions do not "interfere" they could be fired

simultaneously and the choice of firing only one of them at a time becomes an operational rule of the model that hardly relates with the actual characteristics of the system we are modelling.

Table 5. Reachability Set for the Producer/Consumer GSPN of Fig. 7

Vanishing markings:	$\boldsymbol{m}_1$	= (0,1,0,2,0,1)
	$\boldsymbol{m}_2$	= (0,1,0,2,1,0)
	$\boldsymbol{m}_3$	= (1,0,1,1,1,0)
	$\boldsymbol{m}_4$	= (0,1,1,1,0,1)
	$\boldsymbol{m}_5$	= (1,0,2,0,1,0)
	$\boldsymbol{m}_6$	= (0,1,2,0,1,0)
Tangible markings:	$\boldsymbol{m}_7$	= (1,0,0,2,0,1)
	$\boldsymbol{m}_8$	= (1,0,0,2,1,0)
	$\boldsymbol{m}_9$	= (1,0,1,1,0,1)
	$\boldsymbol{m}_{10}$	= (1,0,2,0,0,1)
	$\boldsymbol{m}_{11}$	= (0,1,2,0,0,1)

Assuming that the GSPN is not confused, the computation of the ECSs of the net corresponds to partitioning the set of immediate transitions into equivalence classes such that transitions of the same partition may be in conflict among each other in possible markings, while transitions of different ECSs behave in a truly concurrent manner. When transitions belonging to the same ECS are the only ones enabled in a given marking, one of them (say transition t_k) is selected to fire with probability:

$$P\{t_k|\boldsymbol{m}_i\} = \frac{w_k}{\omega_k(\boldsymbol{m}_i)} \tag{28}$$

where $\omega_k(\boldsymbol{m}_i)$ is the weight of ECS(t_k) in marking $\boldsymbol{m}_i$ and is defined as follows:

$$\omega_k(\boldsymbol{m}_i) = \sum_{t_j \in [\mathrm{ECS}_{(t_k)} \wedge E(\boldsymbol{m}_i)]} w_j \tag{29}$$

Within the ECS we may have transitions that are in direct as well as in indirect conflicts. This means that the firing selection probabilities may be different for the same transition in different markings. Equation (28) however ensures that if we have two transitions (say transitions t_i and t_j), both enabled in two different markings (say markings $\boldsymbol{m}_r$ and $\boldsymbol{m}_s$), the ratios between the firing probabilities of these two transitions in these two markings remain constant and in particular equal to the ratio between the corresponding weights assigned at the moment of the specification of the model.

6.3 Numerical Solution of GSPN Systems

The stochastic process associated with a k-bounded GSPN with $\boldsymbol{m}_0$ as its initial marking (home state) can be classified as a finite state space, stationary (homogeneous), irreducible, and continuous-time semi-Markov process.

Semi-Markov processes can be analysed identifying an embedded (discrete-time) Markov chain that describes the transitions from state to state of the process. In the case of GSPNs, the embedded Markov chain (EMC) can be recognized disregarding the concept of time and focusing the attention on the set of states of the semi-Markov process. The specifications of a GSPN are sufficient for the computation of the transition probabilities of such a chain.

Let RS, TS, and VS indicate the state space (the reachability set), the set of tangible states (or markings) and the set of vanishing markings of the stochastic process, respectively. The following relations hold among these sets:

$$RS = TS \bigcup VS, \qquad TS \bigcap VS = \emptyset.$$

The transition probability matrix $\boldsymbol{U}$ of the EMC can be obtained from the specification of the model using the following expression:

$$u_{ij} = \frac{\sum_{T_k \in E_j(\boldsymbol{m}_i)} w_k}{q_i} \tag{30}$$

In this way, except for the diagonal elements of matrix $\boldsymbol{U}$, all the other transition probabilities of the EMC can be computed using Equation (27) independently of whether the transition to be considered is timed or immediate.

By ordering the markings so that the vanishing ones correspond to the first entries of the matrix and the tangibles to the last, the transition probability matrix $\boldsymbol{U}$ can be decomposed in the following manner:

$$\boldsymbol{U} = \boldsymbol{A} + \boldsymbol{B} = \begin{bmatrix} \boldsymbol{C} & \boldsymbol{D} \\ \boldsymbol{0} & \boldsymbol{0} \end{bmatrix} + \begin{bmatrix} \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{E} & \boldsymbol{F} \end{bmatrix} \tag{31}$$

The elements of matrix $\boldsymbol{A}$ correspond to changes of markings induced by the firing of immediate transitions; in particular, those of submatrix $\boldsymbol{C}$ are the probabilities of moving from vanishings to vanishings, while those of $\boldsymbol{D}$ correspond to transitions from vanishings to tangibles. Similarly, the elements of matrix $\boldsymbol{B}$ correspond to changes of markings caused by the firing of timed transitions: $\boldsymbol{E}$ accounts for the probabilities of moving from tangibles to vanishings, while $\boldsymbol{F}$ comprises the probabilities of remaining within tangible markings.

Indicating with $\boldsymbol{\psi}(n)$ the probability distribution of the EMC at step n (i.e., after n (state-) transitions performed by the EMC), we can compute this quantity using the following expression

$$\boldsymbol{\psi}(n) = \boldsymbol{\psi}(0)\boldsymbol{U}^n \tag{32}$$

where, as usual, $\boldsymbol{\psi}(0)$ represents the initial distribution of the EMC. The steady-state probability distribution $\boldsymbol{\psi}$ can be obtained as the solution of the system of linear equations

$$\begin{cases} \boldsymbol{\psi} \;=\; \boldsymbol{\psi}\, \boldsymbol{U} \\ \boldsymbol{\psi}\, \mathbf{1}^{\mathrm{T}} \;=\; 1 \end{cases} \tag{33}$$

The steady-state probability distribution of the EMC, can be interpreted in terms of numbers of (state-) transitions performed by the EMC. In fact, $1/\psi_i$ is the mean recurrence time for state s_i (marking $\boldsymbol{m}_i$) measured in number of transition firings. The steady-state probability distribution of the stochastic process associated with the GSPN system is thus obtained by weighting each entry ψ_i with the sojourn time of its corresponding marking SJ_i and by normalizing the whole distribution.

Applying these considerations to the Producer/Consumer problem that we are using as an example, the EMC is presented in Table 6.

Table 6. EMC for the Producer/Consumer GSPN of Fig. 7

C =

	1	2	3	4	5	6
1						
2			1			
3						
4						
5						
6				1		

D =

	7	8	9	10	11
1			1		
2					
3	1				
4				1	
5			1		
6					

E =

	1	2	3	4	5	6
7	β					
8		1				
9			α	β		
10					α	
11						1

F =

	7	8	9	10	11
7		α			
8					
9					
10					β
11					

The solution method outlined so far, is computationally acceptable whenever the size of the set of vanishing markings is small (compared with the size of the set of tangible markings). However, this method requires the computation of the steady-state probability of each vanishing marking that is known a priori to be null.

In order to restrict the solution to quantities directly related with the computation of the transient and steady-state probabilities of tangible markings, we must reduce the model by computing the total transition probabilities among tangible markings only, thus identifying a Reduced EMC (REMC).

To illustrate the method of reducing the EMC by removing the vanishing markings, consider first the example of Fig. 21. This system contains two free-choice conflicts corresponding to transitions T_1 , T_2, and t_1, t_2, respectively. From the initial marking $\boldsymbol{m}_i$ with one token in p_1, the system can move to marking $\boldsymbol{m}_j$ (the token in p_3) following two different paths. The first corresponds to the firing of transition T_1, that happens with probability $\frac{\mu_1}{(\mu_1+\mu_2)}$, and that leads to the desired (target) marking $\boldsymbol{m}_j$ in one step only. The second corresponds to selecting transition T_2 to fire first, followed by transition t_1. The first of these two events happens with probability $\frac{\mu_2}{(\mu_1+\mu_2)}$, and the second with probability $\frac{\alpha}{(\alpha+\beta)}$. The total probability of this second path from $\boldsymbol{m}_i$ to $\boldsymbol{m}_j$ amounts to $\frac{\mu_2}{(\mu_1+\mu_2)}\frac{\alpha}{(\alpha+\beta)}$. Notice that firing T_2 followed by t_2 would lead to a different marking (in this case the initial one). Firing T_2 leads the system into an intermediate (vanishing) marking $\boldsymbol{m}_r$. The total probability of moving from $\boldsymbol{m}_i$ to $\boldsymbol{m}_j$ is thus:

$$u'_{ij} = \frac{\mu_1}{(\mu_1+\mu_2)} + \frac{\mu_2}{(\mu_1+\mu_2)}\frac{\alpha}{(\alpha+\beta)} \tag{34}$$

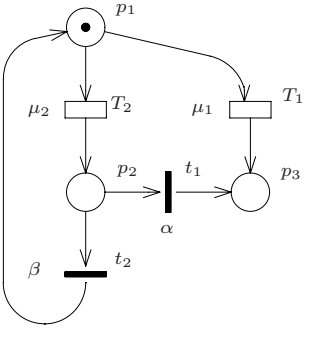


Fig. 21. A GSPN system with multiple paths between tangible markings

In general, recalling the structure of the $\boldsymbol{U}$ matrix, a direct move from marking $\boldsymbol{m}_i$ to marking $\boldsymbol{m}_j$ corresponds to a non-zero entry in block $\boldsymbol{F}$ ($f_{ij} \neq 0$), while a path from $\boldsymbol{m}_i$ to $\boldsymbol{m}_j$ via two intermediate vanishing markings corresponds to the existence of

1. a non-zero entry in block $\boldsymbol{E}$ corresponding to a move from $\boldsymbol{m}_i$ to a generic intermediate marking $\boldsymbol{m}_r$;
2. a non-zero entry in block $\boldsymbol{C}$ from this generic state $\boldsymbol{m}_r$ to another arbitrary vanishing marking $\boldsymbol{m}_s$;
3. a corresponding non-zero entry in block $\boldsymbol{D}$ from $\boldsymbol{m}_s$ to $\boldsymbol{m}_j$.

These informal arguments are precisely captured by the following expression:

$$u'_{ij} \;=\; f_{ij} \;+\; \sum_{r:\boldsymbol{m}_r \in VS} e_{ir} P\{r \to s\}\; d_{sj} \tag{35}$$

where $P\{r \to s\}\; d_{sj}$ is the probability that the net moves from vanishing marking $\boldsymbol{m}_r$ to tangible marking $\boldsymbol{m}_j$ in an arbitrary number of steps, following a path through vanishing markings only.

In order to provide a general and efficient method for the computation of the state transition probability matrix $\boldsymbol{U}'$ of the REMC, we can observe that Equation (35) can be rewritten in matrix notation in the following form:

$$\boldsymbol{U}' \;=\; \boldsymbol{F} \;+\; \boldsymbol{E}\,\boldsymbol{G}\,\boldsymbol{D} \tag{36}$$

where each entry g_{rs} of matrix $\boldsymbol{G}$ represents the probability of moving between vanishing markings $\boldsymbol{m}_r$ and $\boldsymbol{m}_s$ in any number of steps, but without hitting any intermediate tangible marking. $\boldsymbol{G}$ can be expressed in the following way:

$$\boldsymbol{G} \;=\; \sum_{n=0}^{\infty} \boldsymbol{C}^n$$

In the computation of $\boldsymbol{C}^n$, two possibilities may arise. The first corresponds to the situation in which there are no loops among vanishing markings. This means that for any vanishing marking $\boldsymbol{m}_r \in VS$ there is a value n_{0r} such that any sequence of transition firings of length $n \geq n_{0r}$ starting from such marking must reach a tangible marking $\boldsymbol{m}_j \in TS$. In this case

$$\exists n_0 \;:\quad \forall\; n \geq n_0 \quad \boldsymbol{C}^n \;=\; 0$$

and

$$\boldsymbol{G} \;=\; \sum_{k=0}^{\infty} \boldsymbol{C}^k \;=\; \sum_{k=0}^{n_0} \boldsymbol{C}^k$$

The second corresponds to the situation in which there are possibilities of loops among vanishing markings, so that the GSPN may remain "trapped" within a set of vanishing markings. In this case the irreducibility property of the semi-Markov process associated with the GSPN system ensures that the following results hold [51]:

$$\lim_{n\to\infty} \boldsymbol{C}^n \;=\; 0$$

so that

$$\boldsymbol{G} \;=\; \sum_{k=0}^{\infty} \boldsymbol{C}^k \;=\; [\boldsymbol{I} \;-\; \boldsymbol{C}]^{-1}.$$

We can thus write (see [5,4] for details):

$$\boldsymbol{H} \;=\; \begin{cases} \left(\sum_{k=0}^{n_0} \boldsymbol{C}^k \right) \boldsymbol{D} & \textit{no loops among vanishing states} \\ [\boldsymbol{I} \;-\; \boldsymbol{C}]^{-1}\, \boldsymbol{D} & \textit{loops among vanishing states} \end{cases}$$

The transition probability matrix of the REMC can thus be expressed as

$$\boldsymbol{U}' = \boldsymbol{F} + \boldsymbol{E}\,\boldsymbol{H} \tag{37}$$

Once the matrix $\boldsymbol{U}'$ is constructed, standard techniques are used to compute the stationary probability distribution of the REMC and subsequently that of the tangible markings of the GSPN, using their (known) sojourn times.

The computation of the performance indices defined over GSPN models can be performed using the reward method discussed in Section 5 without any additional difficulty.

The advantage of solving the system by first identifying the REMC is twofold. First, the time and space complexity of the solution is reduced in most cases, since the iterative methods used to solve the system of linear equations tend to converge more slowly when applied with sparse matrices and an improvement is obtained by eliminating the vanishing states thus obtaining a denser matrix [24,15]. Second, by decreasing the impact of the size of the set of vanishing states on the complexity of the solution method, we are allowed a greater freedom in the explicit specification of the logical conditions of the original GSPN, making it easier to understand.

The construction of the REMC for the GSPN model of the Producer/Consumer problem is summarized i7 Table 6, where we have exploited the fact that $\boldsymbol{C}^2 = 0$.

The method outlined in this section exploits the elegant mathematical structure of the problem to overcome the difficulties due to the presence of loops of

Table 7. REMC for the Producer/Consumer GSPN of Fig. 7

$$\mathrm{G} = \mathrm{I} + \mathrm{C} =$$

	1	2	3	4	5	6
1	1					
2		1	1			
3			1			
4				1		
5					1	
6				1		1

$$\mathrm{U'} = \mathrm{F{+}EGD} =$$

	7	8	9	10	11
7		α	β		
8	1				
9	α			β	
10			α		β
11				1	

immediate transitions. The loops considered in this derivation are of the "transient" type [24] and correspond to situations in which a steady-state analysis of the model is possible. The REMC is instead impossible to construct following this approach when the loop of immediate transitions is of the "absorbing" type so that during its evolution the net can be trapped into a situation from which it cannot exit. Except for very pathological cases [24] in which the model makes sense despite the presence of such absorbing loops, GSPNs of this type are considered non-well behaving and their analysis is stopped once the existence of absorbing loops of immediate transitions is discovered during the construction of the infinitesimal generator of the REMC. The interested reader is referred to [6] for a detailed discussion of the computational difficulties of the solution and for the many methods that have been developed for handling complex GSPN models of real systems.

7 Conclusions

In this paper we have shown how GSPNs can be conveniently used for the analysis of complex models of DEDS and for their performance and reliability evaluation.

The advantage of net-based models, however, goes far beyond the modelling power of the formalism. In fact, the analysis of the structure of the graph and the computation of its algebraic properties provide information such as invariant conditions, flow-balance equations, and special structures of the reachability graph that can be used to identify models with peculiar solution characteristics, to optimize the solution techniques, and to develop approximation methods.

The practical relevance of GSPNs also highlights a whole set of new problems since larger and larger models are being built and need to be analyzed. Dealing with large models is obviously difficult since even in the case of bounded nets, the size of their reachability sets can become enormous, making their numerical evaluation impossible and discrete-event simulation extremely expensive.

Many important results have been developed in the GSPN field since the time of the introduction of this formalism. In our view, the most important ones are those using the net structure of the model to ease the modelling effort and to improve the efficiency of the solution methods. This allows the analyst to reason about the system at the net level while hiding the complexity of the underlying probabilistic structure.

This has been successfully achieved in dealing with the automatic exploitation of symmetries and modularity, and in incorporating timed transitions with non-exponential distribution that are either deterministic or of phase-type [44].

Symmetries - When dealing with complex systems, it often happens that their models can be constructed via the replication of many identical submodels. To deal with this problem, coloured Petri nets have been proposed to allow the construction of more compact representations [35].

A special class of coloured Petri nets is that of *Stochastic Well Formed Petri Nets* (SWNs) in which restrictions are introduced on the functions that regulate

transition firings and colour manipulations [30,20,19]. The important feature of SWNs is that the special form of their colour functions allows the direct construction of an aggregated state space. With this formalism the symmetries intrinsic in the model are directly exploited to identify markings that are representative of large groups of states having similar characteristics. The aggregation method is fully automated and the direct generation of the aggregated Markov chain is obtained with considerable saving at the level of memory requirements. A significant advantage is also obtained when we must resort to simulation [21].

Block Structure - When designing complex systems we often rely on the definition of components and on their composition for providing advanced services. In these cases, the compositional approach can also be used at the evaluation level when the analyst must maintain a local view of the different components avoiding the direct representation of global system features that, resulting from the interaction of the individual parts, could exhibit unexpected behaviours in pathological cases that are difficult to foresee. The Markov chain that underlies the entire model is only formally specified in this approach in terms of the Markov chains (transition probability matrices) of the individual submodels and of certain correcting factors that account for the interactions among the submodels. The complete transition probability matrix is never really constructed thereby allowing the solution of extremely large models in a very efficient manner. This solution technique also has the non-trivial advantage of being quite suitable for parallelization. In addition, the solution of the entire model is made easier when the submodels interact in very special ways [16], thus making the correcting factors extremely simple.

General Distributions - The assumption of the negative exponential distribution of firing times was soon felt to be too restrictive for a convenient representation of complex systems and several attempts were made to introduce general firing time distributions into the formalism. When generalizing to non exponential distributions the memory policies that we briefly discussed when we introduced the concept of time in the basic PN formalism become extremely important.

Firing delays that are characterized by phase-type distributions were introduced by employing suitable subnet structures that could be easily embedded into GSPN models [17] or by enriching the solution algorithms with proper techniques suited for exploiting their regular structure [28,50].

Using the approach of identifying an embedded Markov chain, a technique has been proposed for the analysis of deterministic and stochastic Petri nets (DSPNs) [8]. In this case the embedded chain is used for the computation of the steady-state solution of nets in which at most one transition of constant delay is enabled in any marking. The transient analysis of DSPN models is presented in [22].

Recently, the class of DSPN models has been extended allowing the transitions to have generally distributed firing times, provided that the constraint of having at most one of these transitions enabled in each marking is still satisfied

[33]. This class of models is also called Markov Regenerative SPNs [36,23] and special formulas for the computation of their steady-state solution were derived. A systematic study of this class of models can be found in [25].

Despite these results that we have briefly overviewed, the computational complexity of the solution of realistic GSPN models remains a difficulty that challanges the effort of many researchers. On one side, there is general consensus that the only means of successfully dealing with large models is to keep them simple by using a "divide and conquer" approach in which the solution of the entire model is constructed on the basis of the solutions of its individual components. A different approach is that of working at another level of abstraction by introducing new components of the fluid type where the details of individual elements are neglected and their overall behaviour is captured by continuous variables.

We believe that, driven by the needs of evaluating more and more compkex systems, many new results will appear in the literature in the near future coming from the above two lines of investigation and contributing to the development of the solid theoretical framework of model construction and analysis.

References

1. T. Agerwala. Putting Petri nets to work. *IEEE Computer*, pages 85–94, December 1979.
2. M. Ajmone Marsan, G. Balbo, A. Bobbio, G. Chiola, G. Conte, and A. Cumani. The effect of execution policies on the semantics and analysis of stochastic Petri nets. *IEEE Transactions on Software Engineering*, 15(7):832–846, July 1989.
3. M. Ajmone Marsan, G. Balbo, G. Chiola, and G. Conte. Generalized stochastic Petri nets revisited: Random switches and priorities. In *Proc. Intern. Workshop on Petri Nets and Performance Models*, pages 44–53, Madison, WI, USA, August 1987. IEEE-CS Press.
4. M. Ajmone Marsan, G. Balbo, and G. Conte. A class of generalized stochastic Petri nets for the performance analysis of multiprocessor systems. *ACM Transactions on Computer Systems*, 2(1), May 1984.
5. M. Ajmone Marsan, G. Balbo, and G. Conte. *Performance Models of Multiprocessor Systems*. MIT Press, Cambridge, USA, 1986.
6. M. Ajmone Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis. *Modelling with Generalized Stochastic Petri Nets*. J. Wiley, 1995.
7. M. Ajmone Marsan, G. Balbo, and K.S. Trivedi, editors. *Proc. Intern. Workshop on Timed Petri Nets*, Torino, Italy, July 1985. IEEE-CS Press.
8. M. Ajmone Marsan and G. Chiola. On Petri nets with deterministic and exponential transition firing times. In *Proc. 7^{th} European Workshop on Application and Theory of Petri Nets*, pages 151–165, Oxford, England, June 1986.
9. H. Alaiwan and G. Memmi. Algorithmes de recherche des solutions entieres positives d'un systeme d'equations lineaires homogeneus en nombres entieres. *Revue Technique Thomson-CSF*, 14(1):125–135, March 1982. in French.
10. H. Alaiwan and J. Mt Toudic. Research des semiflots, des verrous et des trappes dans le reseaux de Petri. *Technique et Science Informatiques*, 4(1), February 1985.
11. G. Balbo. Exponential stochastic petri nets. In G. Balbo and M. Silva, editors, *Performance Models for Discrete Event Systems with Synchronisations: Formalisms and Analysis Techniques*, pages 307–344. KRONOS, Zaragoza, Spain, 1998.

12. G. Balbo. Non-exponential stochastic petri nets. In G. Balbo and M. Silva, editors, *Performance Models for Discrete Event Systems with Synchronisations: Formalisms and Analysis Techniques*, pages 345–385. KRONOS, Zaragoza, Spain, 1998.
13. G. Balbo. Introduction to stochastic petri nets. In E. Brinksma, H. Hermanns, and J.-P. Katoen, editors, *Formal Methods and Performance Analysis, LNCS Vol. 2090*, pages 84–155. Springer-Verlag, Berlin, Germany, May 2001.
14. G. Balbo, M. Silva, and editors. *Performance Models for Discrete Event Systems with Synchronizations: Formalisms and Analysis Techniques*. KRONOS, Zaragoza, Spain, 1998.
15. A. Blakemore. The cost of eliminating vanishing markings from generalized stochastic Petri nets. In *Proc.* 3^{rd} *Intern. Workshop on Petri Nets and Performance Models*, Kyoto, Japan, December 1989. IEEE-CS Press.
16. P. Buchholz. Hierarchical high level Petri nets for complex system analysis. In *Proc.* 15^{th} *Intern. Conference on Applications and Theory of Petri Nets*, number 185 in LNCS, Zaragoza, Spain, 1994. Springer-Verlag.
17. P. Chen, S.C. Bruell, and G. Balbo. Alternative methods for incorporating non-exponential distributions into stochastic Petri nets. In *Proc.* 3^{rd} *Intern. Workshop on Petri Nets and Performance Models*, pages 187–197, Kyoto, Japan, December 1989. IEEE-CS Press.
18. G. Chiola, M. Ajmone Marsan, G. Balbo, and G. Conte. Generalized stochastic Petri nets: A definition at the net level and its implications. *IEEE Transactions on Software Engineering*, 19(2):89–107, February 1993.
19. G. Chiola, C. Dutheillet, G. Franceschinis, and S. Haddad. Stochastic Well-Formed coloured nets for symmetric modelling applications. *IEEE Transactions on Computers*, 42(11), November 1993.
20. G. Chiola and G. Franceschinis. Colored GSPN models and automatic symmetry detection. In *Proc.* 3^{rd} *Intern. Workshop on Petri Nets and Performance Models*, Kyoto, Japan, December 1989. IEEE-CS Press.
21. G. Chiola, G. Franceschinis, R. Gaeta, and M. Ribaudo. GreatSPN1.7: GRaphical Editor and Analyzer for Timed and Stochastic Petri Nets. *Performance Evaluation*, 24, 1995. Special Issues on Performance Modelling Tools.
22. H. Choi, G. Kulkarni, and K.S. Trivedi. Transient analysis of deterministic and stochastic petri nets. In *Proc.* 14^{th} *Intern. Conference on Application and Theory of Petri Nets*, Chicago, Illinois, June 1993. Springer Verlag.
23. H. Choi, V.S. Kulkarni, and K.S. Trivedi. Markov regenerative stochastic Petri nets. In *Proc. of Performance 93*, Rome, Italy, September 1993.
24. G. Ciardo. *Analysis of large Petri net models*. PhD thesis, Department of Computer Science, Duke University, Durham, NC, USA, 1989. Ph. D. Thesis.
25. G. Ciardo, R. German, and C. Lindemann. A characterization of the stochastic process underlying a stochastic Petri net. In *Proc.* 5^{th} *Intern. Workshop on Petri Nets and Performance Models*, pages 170–179, Toulouse, France, October 1993. IEEE-CS Press.
26. G. Ciardo and C. Lindemann. Analysis of deterministic and stochastic Petri nets. In *Proc.* 5^{th} *Intern. Workshop on Petri Nets and Performance Models*, pages 160–169, Toulouse, France, October 1993. IEEE-CS Press.
27. G. Ciardo, J. Muppala, and K.S. Trivedi. On the solution of GSPN reward models. *Performance Evaluation*, 12(4):237–253, July 1991.
28. A. Cumani. On the canonical representation of homogeneousMarkov processes modelling failure time distributions. *Microelectron Reliability*, 22:583–602, 1982.

29. J.B. Dugan, K.S. Trivedi, R.M. Geist, and V.F. Nicola. Extended stochastic Petri nets: Applications and analysis. In *Proc. PERFORMANCE '84*, Paris, France, December 1984.
30. C. Dutheillet and S. Haddad. Aggregation and disaggregation of states in colored stochastic Petri nets: Application to a multiprocessor architecture. In *Proc.* 3^{rd} *Intern. Workshop on Petri Nets and Performance Models*, Kyoto, Japan, December 1989. IEEE-CS Press.
31. G. Florin and S. Natkin. Les reseaux de Petri stochastiques. *Technique et Science Informatiques*, 4(1), February 1985.
32. G. Florin and S. Natkin. Matrix product form solution for closed synchronized queueing networks. In *Proc.* 3^{rd} *Intern. Workshop on Petri Nets and Performance Models*, pages 29–39, Kyoto, Japan, December 1989. IEEE-CS Press.
33. R. German and C. Lindemann. Analysis of stochastic Petri nets by the method of supplementary variables. In *Proc. of Performance 93*, Rome, Italy, September 1993.
34. K. Lautenbach. Linear algebraic technique for place/transition nets. In W. Brawer, W. Reisig, and G. Rozenberg, editors, *Advances on Petri Nets '86 - Part I*, volume 254 of *LNCS*, pages 142–167. Springer Verlag, Bad Honnef, West Germany, February 1987.
35. C. Lin and D.C. Marinescu. On stochastic high level Petri nets. In *Proc. Intern. Workshop on Petri Nets and Performance Models*, Madison, WI, USA, August 1987. IEEE-CS Press.
36. V. Mainkar, H. Choi, and K. Trivedi. Sensitivity analysis of Markov regenerative stochastic Petri nets. In *Proc.* 5^{th} *Intern. Workshop on Petri Nets and Performance Models*, Toulouse, France, October 1993. IEEE-CS Press.
37. J. Martinez and M. Silva. A simple and fast algorithm to obtain all invariants of a generalized Petri net. In *Proc.* 2^{nd} *European Workshop on Application and Theory of Petri Nets*, Bad Honnef, West Germany, September 1981. Springer Verlag.
38. P. M. Merlin and D. J. Farber. Recoverability of communication protocols: Implications of a theoretical study. *IEEE Transactions on Communications*, 24(9):1036–1043, September 1976.
39. J. F. Meyer, A. Movaghar, and W. H. Sanders. Stochastic activity networks: Structure, behavior, and application. In *Proc. Intern. Workshop on Timed Petri Nets*, pages 106–115, Torino,Italy, July 1985.
40. M.K. Molloy. *On the Integration of Delay and Throughput Measures in Distributed Processing Models*. PhD thesis, UCLA, Los Angeles, CA, 1981. Ph.D. Thesis.
41. M.K. Molloy, T. Murata, and M.K. Vernon, editors. *Proc. Intern. Workshop on Petri Nets and Performance Models*, Madison, Wisconsin, August 1987. IEEE-CS Press.
42. T. Murata. Petri nets: properties, analysis, and applications. *Proceedings of the IEEE*, 77(4):541–580, April 1989.
43. S. Natkin. *Les Reseaux de Petri Stochastiques et leur Application a l'Evaluation des Systemes Informatiques*. PhD thesis, CNAM, Paris, France, 1980. These de Docteur Ingegneur.
44. M.F. Neuts. *Matrix Geometric Solutions in Stochastic Models*. Johns Hopkins University Press, Baltimore, MD, 1981.
45. J. D. Noe and G. J. Nutt. Macro e-nets representation of parallel systems. *IEEE Transactions on Computers*, 31(9):718–727, August 1973.
46. J.L. Peterson. *Petri Net Theory and the Modeling of Systems*. Prentice-Hall, Englewood Cliffs, NJ, 1981.

47. C.A. Petri. Communication with automata. Technical Report RADC-TR-65-377, Rome Air Dev. Center, New York, NY, 1966.
48. W. Reisig. *Petri Nets: an Introduction*. Springer Verlag, 1985.
49. F. J. W. Symons. Introduction to numerical Petri nets, a general graphical model of concurrent processing systems. *Australian Telecommunications Research*, 14(1):28–33, January 1980.
50. M. Telek and A. Bobbio. Markov regenerative stochastic petri nets with age type general transitions. In *Proc. 16th International Conference on Application and Theory of Petri Nets*, LNCS. Springer Verlag, 1995.
51. R.S. Varga. *Matrix Iterative Analysis*. Prentice-Hall, Englewood Cliffs, NJ, 1962.

Stochastic Process Algebras

Allan Clark, Stephen Gilmore, Jane Hillston, and Mirco Tribastone

LFCS, School of Informatics, University of Edinburgh

Abstract. In this tutorial we give an introduction to stochastic process algebras and their use in performance modelling, with a focus on the PEPA formalism. A brief introduction is given to the motivations for extending classical process algebra with stochastic times and probabilistic choice. We then present an introduction to the modelling capabilities of the formalism and the tools available to support Markovian based analysis. The chapter is illustrated throughout by small examples, demonstrating the use of the formalism and the tools.

1 Introduction

Process algebras emerged as a modelling technique for the functional analysis of concurrent systems approximately twenty years ago. Over the last 17 years there have been several attempts to take advantage of the attractive features of this modelling paradigm within the field of performance evaluation.

Stochastic process algebras (SPA) were first proposed as a tool for performance and dependability modelling in 1990 [1]. At that time there was already a plethora of techniques for constructing performance models so the introduction of another one could have been deemed unnecessary if it were not for the fact that SPA offered something new—formally defined compositionality. Queueing networks, which have been widely used for performance modelling for more than thirty years, have an inherent compositionality but this is implicit and informal. Stochastic extensions of Petri nets have a semantic model but, in general, no clear compositional structure. In the process algebra the compositionality is explicit—provided by the combinators of the language—and formal—supported by the semantics and equivalence relations of the language.

It was immediately clear that having this explicit structure within models offers benefits for model construction:

- when a system consists of interacting components, the components, and the interaction, can each be modelled separately;
- models have a clear structure and are easy to understand;
- models can be constructed systematically, by either elaboration or refinement;
- the possibility of maintaining a library of model components, supporting model reusability, is introduced.

Many case studies demonstrating these and other benefits have appeared in the literature [2,3,4,5,6,7,8,9,10,11,12,13,14,15,16].

M. Bernardo and J. Hillston (Eds.): SFM 2007, LNCS 4486, pp. 132–179, 2007.

A limitation of the initial SPA languages was their lack of expressiveness with respect to timing distributions. Essentially, they restricted consideration to models in which all durations were represented by negative exponentially distributed random variables. Some later work has aimed to change this situation by considering languages in which generally distributed random variables may be associated with the actions of a model. However such models are not so amenable to quantitative analysis and therefore their practical uptake has been limited.

The remainder of this tutorial is organised as follows. In the following section we present a short introduction to classical process algebras as they are used for system verification from a functional or qualitative point of view. Stochastic process algebras generally, and the language PEPA specifically, are presented in Section 3. Section 4 describes model analysis. The tools available to support the approaches we have described are discussed in Section 5, and we present some case studies in the following section. In Section 7 we continue to advanced topics such as continuous state-space approximation.

2 Classical Process Algebras

Process algebras are abstract languages used for the specification and design of concurrent systems. The most widely known process algebras are Milner's Calculus of Communicating Systems (CCS) [17] and Hoare's Communicating Sequential Processes (CSP) [18]. The stochastic process algebras take inspiration from both these formalisms. Models in CCS and CSP have been used extensively to establish the correct behaviour of complex systems by deriving *qualitative* properties such as *freedom from deadlock* or *livelock*.

In the process algebra approach systems are modelled as collections of entities, called *agents*, which execute atomic *actions*. These actions are the building blocks of the language and they are used to describe sequential behaviours which may run concurrently, and synchronisations or communications between them.

In CCS two agents communicate when one performs an action, a say, while the other performs the complementary action $\bar{a}$. The resulting communication action has the distinguished label τ, which represents an *internal* action that is invisible to the environment. Agents may proceed with their internal actions simultaneously but it is important to note that the semantics given to the language imposes an interleaving on such concurrent behaviour. The basic calculus contains the following primitives for defining agents:

prefix	$a.B$	after action a the agent becomes B
parallel composition	$A \vert B$	agents A and B proceed in parallel
choice	$A + B$	the agent behaves as A or B depending on which acts first

restriction	$A\backslash M$	the set of labels M is hidden from outside agents
relabelling	$A[a_1/a_0, ..]$	in this agent label a_1 is renamed a_0
the null agent	**0**	this agent cannot act (deadlock)

The communication mechanism in CSP is different as there is no notion of complementary actions: this is a major distinction between CCS and CSP. In CSP two agents communicate by simultaneously executing actions with the same label. Since during the communication the joint action remains visible to the environment, it can be reused by other concurrent processes so that more than two processes can be involved in the communication (*multiway synchronisation*). This is the communication mechanism adopted by most of the SPA languages.

Like many other process algebras, CCS is given a structured operational semantics (SOS), using a labelled transition system. From this a *derivative tree* or *graph* may be constructed in which language terms form the nodes and transitions are the arcs. This structure is a useful tool for reasoning about agents and the systems they represent. It is also the basis of the *bisimulation* style of equivalence. In this style of equivalence, the actions of an agent characterise it, so two agents are considered to be equivalent if they are observed to perform exactly the same actions. Strong and weak forms of equivalence are defined depending on whether the internal actions of an agent are deemed to be observable.

In CCS and CSP, since the objective is qualitative analysis rather than quantitative, time is abstracted away. Various suggestions for incorporating time into these formalisms have been investigated (see [19] for an overview). For example, Temporal CCS [20] extends CCS with *fixed delays* and *wait-for synchronisation* (asynchronous waiting):

fixed time delay	(t)	the agent must wait t time units before performing its next action
wait-for synchronisation	δ	the agent may idle indefinitely until its next action is possible
non-temporal deadlock	$\underline{0}$	the agent idles indefinitely and never engages in further actions.

Note that most of the timed extensions, including TCCS, retain the assumption that actions are instantaneous and regard time progression as orthogonal to the activity of the system. In contrast, the early SPAs generally associated a random variable, representing duration, with each action. The alternative approach, of separating action and time, is adopted in most of the work incorporating non-exponentially distributed durations.

Similarly process algebras are often used to model systems in which there is uncertainty about the behaviour of a component, but this uncertainty is

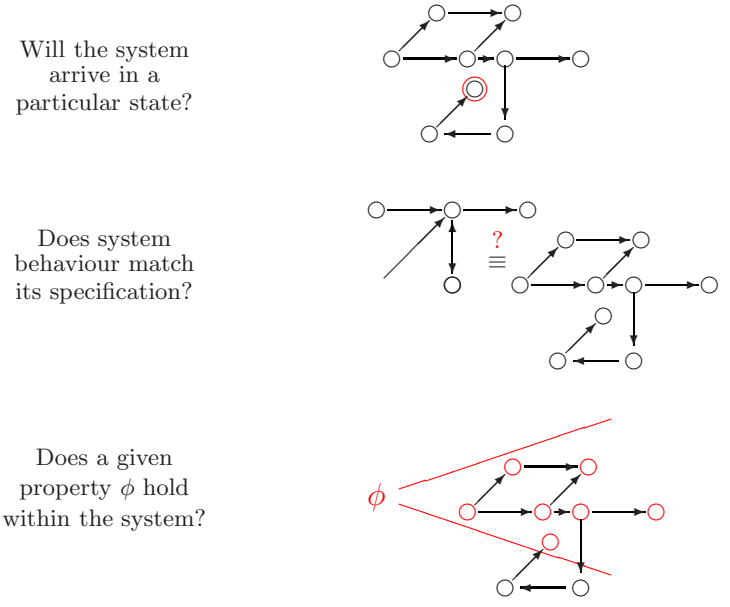


Fig. 1. Functional analysis of process algebra

abstracted away so that all choices become nondeterministic. Probabilistic extensions of process algebras, such as PCCS [21], allow this uncertainty to be quantified using a probabilistic choice combinator. In this case a probability is associated with each possible outcome of a choice. In some SPA an alternative approach is taken—we assume that a *race condition* resolves choices when more than one (timed) action can occur.

The Concurrency Workbench (CWB) [22] is a tool that automates the checking of assertions about CCS models in order to establish properties of the systems they describe. As well as the basic calculus, it supports a synchronous variant and the temporal extension, TCCS. The CWB allows simple properties, such as presence of deadlock, to be checked directly, but needs more specific properties to be expressed in a suitable logic. In the context of process algebra modelling, a process logic is a natural way to frame properties and queries. Such logics, known as *modal logics*, express assertions about changing state. There is a simple modal logic, Hennessy-Milner logic [23], for immediate possibilities in a model, and an extended logic, the modal $\mu-$calculus [24], with fixed point operators for recursive definitions.

3 Stochastic Process Algebra: PEPA

Process algebras offer several attractive features which are not necessarily available in existing performance modelling paradigms. The most important of these are *compositionality*, the ability to model a system as the interaction of its subsystems,

How long will it take for the system to arrive in a particular state?

With what probability does system behaviour match its specification?

Does a given property ϕ hold within the system within time t with probability p?

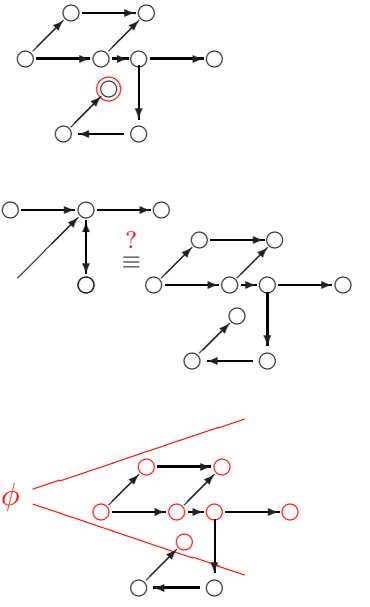


Fig. 2. Quantitative analysis of stochastic process algebra

formality, giving a precise meaning to all terms in the language, and *abstraction*, the ability to build up complex models from detailed components but disregarding internal behaviour when it is appropriate to do so. Queueing networks offer compositionality but not formality; stochastic extensions of Petri nets offer formality but not compositionality; neither offer abstraction mechanisms.

In the early 1990s several stochastic extensions of process algebra appeared in the literature, motivated by a desire to add quantification to process algebra models and make them suitable for performance modelling. These included TIPP [25] from the University of Erlangen, EMPA[1] [26,27] from the University of Bologna, PEPA [28,29] from the University of Edinburgh and SPADE[2] [30] from Imperial College. PEPA was the first language to be developed with the intention of generating Markov processes which could be solved numerically for performance evaluation, but versions of TIPP and EMPA from around the same time are similarly Markovian based. The other Markovian-based SPA, emerged a little later the stochastic π-calculus [31] and IMC [32] and differ in terms of their synchronisation and treatment of delay, respectively. For the remainder of this section, and the following one, we concentrate on PEPA; however, towards the end of this section we will discuss how TIPP and EMPA differ from PEPA. SPADE was developed with a different motivation, relating to generalised semi-Markov processes and simulation. Several other calculi have also

[1] Originally called simply MPA.
[2] Originally called CCS+.

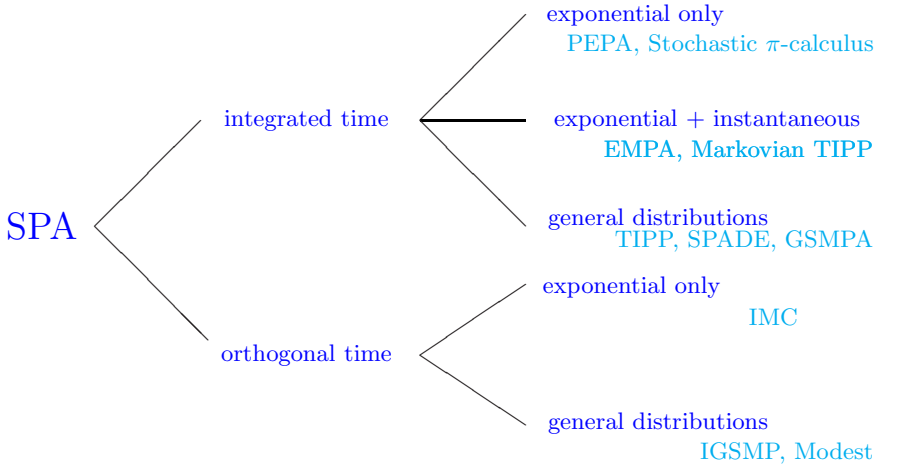


Fig. 3. Classification of the stochastic process algebras

incorporated generally distributed activities or delays, e.g. Modest [33], IGSMP [34] and GSMPA [35].

PEPA (Performance Evaluation Process Algebra) extends classical process algebra by associating a random variable, representing duration, with every action. These random variables are assumed to be exponentially distributed and this leads to a clear relationship between the process algebra model and a continuous time Markov process. Via this underlying Markov process performance measures can be extracted from the model.

PEPA models are described as interactions of *components*. Each component can perform a set of actions: an action $a \in \mathcal{A}ct$ is described as a pair (α, r), where $\alpha \in \mathcal{A}$ is the *type* of the action and $r \in \mathbb{R}^+$ is the parameter of the negative exponential distribution governing its duration. Whenever a process P can perform an action, an instance of a given probability distribution is sampled: the resulting number specifies how long it will take to *complete* the action. A small but powerful set of combinators is used to build up complex behaviour from simpler behaviour. The combinators are familiar from classical process algebra: prefix, choice, parallel composition and abstraction. We explain each of the combinators informally below. A formal operational semantics for PEPA is available in [29].

Prefix: A component may have purely sequential behaviour, repeatedly undertaking one activity after another and eventually returning to the beginning of its behaviour. A simple example is a web service within a distributed system, which can serve one request at a time. Each application requiring the web service will need to gain access to the service which will then only be made available for another application when a response has been successfully transferred.

$$WS \stackrel{\text{def}}{=} (request, \top).(serve, \mu).(respond, \top).WS$$

In some cases, as here, the rate of an action is outside the control of this component. Such actions are carried out jointly with another component, with this component playing a passive role. For example, the web service is passive with respect to the *request* action, as it cannot influence the rate at which requests arrive, and this is recorded by the distinguished symbol, $\top$ (called "top").

Choice: A choice between two possible behaviours is represented as the sum of the possibilities. For example, if we consider an application in a distributed system, a computation may have two possible outcomes: access to a locally available method is required (with probability p_1) or access to a remote web service is necessary (with probability $p_2 = 1 - p_1$). In this example the *think* action denotes processing within the application. These alternatives are represented as shown below:

$$Appl \stackrel{\text{def}}{=} (think, p_1\lambda).(local, m).Appl \\ + \quad (think, p_2\lambda).(request, rq).(respond, rp).Appl$$

A race condition governs the behaviour of simultaneously enabled actions so the choice combinator represents pre-emptive selection with re-sampling. The continuous nature of the probability distributions ensures that the actions cannot occur simultaneously. Thus a sum will behave as either one summand or the other. When an action has more than one possible outcome, e.g. the *think* action in the application, it is represented by a choice of separate actions, one for each possible outcome. The rates of these actions are chosen to reflect their relative probabilities.

Parallel composition: As mentioned earlier, PEPA and most of the other SPA adopt the parallel composition from CSP, rather than that from CCS. Correspondingly, there is no notion of complementary actions and multiway synchronisations are possible.

In the web service example, we have already anticipated that the application and the web service will be working together within the same system. This will require them to *cooperate* when the application needs the service offered by the web service, which is not available locally. In contrast, the local activities of the application can be carried out independently of the web service. Cooperation over given actions is reflected in the parallel composition by the *cooperation set*, $L = \{request, serve, respond\}$ in this case. Actions in this set require the simultaneous involvement of both components. The resulting action, a *shared* action, will have the same type as the two contributing actions and a rate reflecting the rate of the action in the slowest participating component. Note that this means that the rate of a passive action will become the rate of the action it cooperates with.

If, for simplicity, we assume that the distributed system consists of just two independent applications, the system is represented as the cooperation of the applications and the web service as follows:

$$Sys_1 \stackrel{\text{def}}{=} \left(Appl \parallel Appl\right) \underset{L}{\bowtie} WS \qquad L = \{request, serve, respond\}$$

The combinator $\|$ is a degenerate form of the cooperation combinator, formed when two components behave completely independently, without any cooperation between them, as in the case of the two independent applications. This pure parallel combinator can be thought of as cooperation over the empty set: $(Appl \underset{\emptyset}{\bowtie} Appl)$.

Abstraction: Again, the abstraction mechanism used in SPA follows CSP rather than CCS. It is often convenient to hide some actions, making them private to the component or components involved. The duration of the actions is unaffected, but their type becomes hidden, appearing instead as the unknown type τ. Components cannot synchronise on τ. For example, as we further develop the model of the distributed system we may wish to hide the access of a application to its local method. This might lead to a new representation of the application:

$$Appl' \stackrel{\text{def}}{=} Appl/\{local\}$$

and a corresponding new representation of the system:

$$Sys_2 \stackrel{\text{def}}{=} \left(Appl' \parallel Appl'\right) \underset{L}{\bowtie} WS \qquad L = \{request, serve, respond\}$$

Note that this is quite different from the CCS restriction operator which prevents actions of the given label from occurring.

Use of the hiding combinator has two implications. Firstly, it ensures that no components added to the model at a later stage can invoke this method of the application. Secondly, private actions are deemed to have no contribution to the performance measures being calculated and this might subsequently suggest simplifications to the model.

Throughout the simple example above we have used constants such as WS to associate names with behaviours. Using recursive definitions we have been able to describe components with infinite behaviours without the use of an explicit recursion operator.

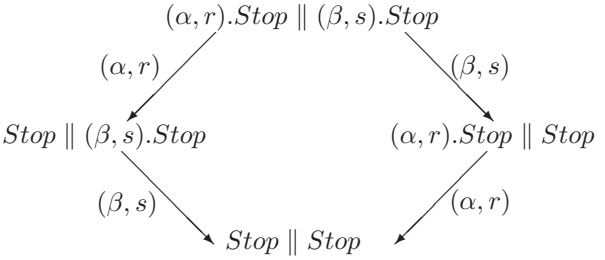
Representing the components of the system as separate components means that we can easily extend our model. Now we may want to consider a distributed system consisting of more than two applications which act independently of each other but compete for the use of web service. To enhance fault tolerance the web service may be replicated. This extension may be achieved compositionally by combining more instances of the components already described. For example, in the case of three applications and two instances of the web service we have:

$$Sys_3 \stackrel{\text{def}}{=} \left(Appl \parallel Appl \parallel Appl\right) \underset{L}{\bowtie} \left(WS \parallel WS\right) \qquad L = \{request, serve, respond\}$$

3.1 Designing the Language

Action durations. The selection of a negative exponential distribution as the governing distribution for the action durations in PEPA and other SPA has profound consequences. In terms of the underlying stochastic process, it is the only choice which gives rise to a Markov process. In terms of the process algebra it

is the only choice which preserves the well-known *expansion law* which underlies the interleaving semantics. In both cases this is due to the *memoryless* property of the exponential distribution: the time until the next event is independent of the time since the last event—the exponential distribution "forgets" how long it has already waited. Thus if we consider a process $(\alpha, r).Stop \parallel (\beta, s).Stop$, from the semantics we derive:



In a generally timed (or even deterministically timed) scenario it would be important to record the elapsed time in the intermediate states in order to know the residual time of the remaining activity. For example, the time needed to complete β in $Stop \parallel (\beta, s).Stop$ should reflect the time already taken to complete activity α. However the memoryless property of the exponential distribution tells us that the distribution of the residual time in β is the same as it was initially in state $(\alpha, r).Stop \parallel (\beta, s).Stop$ before any time had elapsed. Thus we retain the expansion law of classical process algebra:

$$(\alpha, r).Stop \parallel (\beta, s).Stop = \\ (\alpha, r).(\beta, s).(Stop \parallel Stop) + (\beta, s).(\alpha, r).(Stop \parallel Stop)$$

Later formalisms which incorporated general distributions either avoided the issue of residual durations by separating actions and delays (e.g. Modest [33] and IGSMP [34]), or used a finer-grained semantics such as ST-semantics to distinguish the start and stop of each action (e.g. GSMPA [35]).

Another major difference between the SPA formalisms concerns immediate or instantaneous actions. EMPA has immediate actions, modelled after the immediate transitions of GSPN. Each immediate action has an associated priority level and an associated weight. Immediate actions always have higher priority than exponentially timed actions, so a choice between such actions is resolved by priorities. If two immediate actions of the same priority level are concurrently enabled, the choice is resolved on the basis of their associated weights. The inclusion of immediate actions in TIPP, in addition to those with an associated exponentially distributed delay, has also been investigated. These actions were used to model logical [36] or control activities [37]. In these cases it was assumed that the environment of the component will resolve choices, but this opens the possibility that a model may contain non-determinism. Such a model is considered to be under-specified.

Cooperation. Communication or parallel composition is the essence of compositionality in process algebras. It gives structure to models, indicating which actions may be undertaken concurrently, and which cannot.

For most SPA the choice was made to adopt the multiway synchronisation using shared names (as in CSP) rather than complementary actions (as in CCS). This means that components or agents jointly perform actions of the same type, when the parallel composition dictates it. The motivation was to represent something more general than communication. In performance models interaction often captures resource usage and the objective of the model is to study the constraints imposed on components by competition over resources. In this context the multiway synchronisation offered more generality. However this choice was independent of the quantification of action durations, as witnessed by the adoption of CCS-style synchronisation in the stochastic π-calculus which generates a Markov process in the same way as PEPA [31].

Nevertheless the quantification of action duration did pose a challenge for the definition of cooperation. Actions which are to be performed jointly may each have been assigned rates (durations) in their respective components. The best way to resolve what should be the rate of the shared action has been a topic of some debate. The differing solutions adopted have become the main distinguishing feature of the various SPA formalisms.

The first observation is that if we view the joint action as a "synchronisation" as in the sense of *barrier synchronisation* in parallel programming then the correct duration would be the maximum of the durations, i.e. the maximum of the random variables. The unfortunate problem is that the maximum of two or more exponentially distributed random variables is not exponentially distributed.

In PEPA it is assumed that each component has *bounded capacity* to carry out activities of any particular type, determined by the *apparent rate*. For a component P and action type α, the apparent rate of α in P, denoted $r_\alpha(P)$, is the sum of the rates of each α action enabled in P. This corresponds to the rate at which P appears to an external observer to carry out an α action, due to the superposition principle of the negative exponential distribution. The definition of cooperation in PEPA is based on the assumption that a component cannot be made to exceed its bounded capacity, meaning that the apparent rate of the shared action will be the minimum of the apparent rates of the components involved.

In TIPP the "rate" is assumed to represent work capacity in one partner of the synchronisation and work demand in the other. The rate of the shared action is then taken to be the product of the two component rates. In contrast, in EMPA it is assumed that in any synchronisation exactly one participant has an explicit representation for the rate of the activity, all other participants being *passive* with respect to this activity, prepared to proceed at the rate of the active participant. This scheme does satisfy the principle of bounded capacity but the restriction has implications for the compositionality of the language. The formalisms which separate action and time evolution avoid this issue by only allowing synchronisation on untimed actions. The issue of timed synchronisation is discussed in [38,39] and in detail in Bradley's thesis [40].

4 Model Analysis

The formality of the process algebra approach allows us to assign a precise meaning to every language expression. This implies that once we have a language description of a given system its behaviour can be deduced automatically. The meaning, or semantics, of a PEPA expression is provided by SOS rules as for CCS, which associates a labelled multi-transition system with every expression in the language [29].

A labelled transition system $(S, T, \{\xrightarrow{t} \mid t \in T\})$ consists of a set of states S, a set of transition labels T and a transition relation $\xrightarrow{t} \subseteq S \times S$. For PEPA the states are the syntactic terms in the language, the transition labels are the actions $((type, rate)$ pairs), and the transition relation is given by the semantic rules. A multi-transition relation is used because the number of instances of a transition (action) is significant since it can affect the timing behaviour of a component.

Based on the transition relation, a transition diagram, called the *derivation graph* (DG), can be associated with each language expression. This graph describes all the possible evolutions of any component and provides a useful way to reason about the behaviour of a model. A certain amount of care is needed in defining the derivation graph. Consider a simple component, P, which will repeatedly carry out the action $a = (\alpha, r)$, i.e. $P \stackrel{\text{def}}{=} (\alpha, r).P$. For a classical process algebra we need only consider which actions it is possible for an agent to perform. Thus, the agent $P + P$ has the same behaviour as the agent P—both are capable of an α named action and subsequently behave as P—so these agents are considered to be equivalent. In a SPA multiple instances of an action become apparent because the duration of an action of that type will be the minimum of the corresponding random variables, i.e. the apparent rate of the action will be the sum of the rates. Thus $P + P$ appears to carry out the first α named action at twice the rate of the agent P. Consequently the two cannot be regarded as equivalent.

Alternative solutions have been offered for this problem. In TIPP and EMPA supplementary labels are used to distinguish instances of multiply enabled actions, and the underlying structure is still a labelled transition system. In PEPA the semantics of the language is given in terms of a labelled multi-transition system with the transition relation represented as a multi-relation in which the multiplicities of arcs are recorded.

An example derivation graph is shown in Figure 4 where the DG of the PEPA model Sys_4, consisting of a single application accessing the web service, is shown. For didactic purposes, in the left hand part of the figure we have expanded the derivatives of the components *Appl* and WS.

Inspection of the DG allows one to derive qualitative properties of the model. In this case, for instance, we can see that the PEPA model is free from deadlock and live. Moreover, the Markov process underlying any finite PEPA component can be obtained directly from the DG: a state of the Markov process is associated with each node of the graph and the transitions between states are defined by

$$
\begin{aligned}
Appl &\stackrel{\text{def}}{=} (think, p_1\lambda).Appl_1 \\
&\quad + (think, p_2\lambda).Appl_2 \\
Appl_1 &\stackrel{\text{def}}{=} (local, m).Appl \\
Appl_2 &\stackrel{\text{def}}{=} (request, rq).Appl_3 \\
Appl_3 &\stackrel{\text{def}}{=} (respond, rp).Appl \\
\\
WS &\stackrel{\text{def}}{=} (request, \top).WS_1 \\
WS_1 &\stackrel{\text{def}}{=} (serve, \mu).WS_2 \\
WS_2 &\stackrel{\text{def}}{=} (respond, \top).WS \\
\\
Sys_4 &\stackrel{\text{def}}{=} Appl \underset{L}{\bowtie} WS \\
L &= \{request, serve, respond\}
\end{aligned}
$$

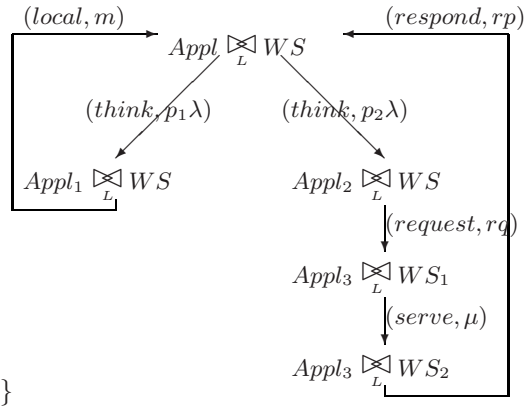


Fig. 4. Derivation graph underlying Sys_4

considering the rates labelling the arcs. Since all activity durations are exponentially distributed, the total transition rate between two states will be the sum of the activity rates labelling arcs connecting the corresponding nodes in the DG. Starting from the DG of Figure 4, the derivation of the corresponding Markov process is straightforward and results in the generator matrix shown below.

$$
\boldsymbol{Q} = \begin{pmatrix}
-\lambda & p_1\lambda & p_2\lambda & 0 & 0 \\
m & -m & 0 & 0 & 0 \\
0 & 0 & -rq & rq & 0 \\
0 & 0 & 0 & -\mu & \mu \\
rp & 0 & 0 & 0 & -rp
\end{pmatrix}
$$

Once obtained, the infinitesimal generator matrix can be used for a variety of different analysis techniques. Most commonly the model is subjected to steady state analysis. This assumes that the Markov process will eventually reach a regular pattern of behaviour and the probability distribution over the states of the model will cease to change, i.e. that the Markov process is *ergodic*. For such models the steady state probability distribution can be derived and reveals much information about the steady state, or equilibrium, behaviour of the model. In addition, any Markov process (both ergodic and not) can be subjected to transient analysis. In its simplest form a transient analysis will derive the state probability distribution for a given starting state and after a given time. However, it is also the basis of more sophisticated analyses such as calculating first passage and response time distributions.

In order to ensure that the Markov process underlying a PEPA model is ergodic, the DG of a PEPA model must be strongly connected. Necessary conditions for ergodicity, at the syntactic level of a PEPA model, have been defined [29]. For example, if cooperation occurs it must be the highest level combinator.

The class of PEPA terms which satisfy these syntactic conditions are termed *cyclic components* and they can be described by the following grammar:

$$P ::= S \mid P \underset{L}{\bowtie} P \mid P/L$$
$$S ::= (\alpha, r).S \mid S + S \mid A$$

All the models we have discussed so far satisfy the syntactic conditions required to be cyclic models.

It is well known that if the Markov process is ergodic, it is possible to compute the steady state probability distribution over all the possible states by solving the matrix equation $\pi \boldsymbol{Q} = 0$ where $\boldsymbol{Q}$ is the generator matrix of the Markov process and π is the state probability vector, such that $\sum_i \pi_i = 1$.

The probability distribution of the states of the model is often not the ultimate goal of performance analysis. Performance measures such as throughput and utilisation are often derived via a *reward structure* which is defined over the Markov process. This can either be done explicitly by the modeller, or as we will see, automatically by the tool for commonly required measures. A reward structure associates a value or *reward* with each state of the model. For steady state measures, the expected value of the reward (i.e. the sum over the entire state space of (probability of a state $\times$ reward in that state)) is calculated. In a process algebra it can be easier to associate rewards with actions. In this case the reward associated with a state will be the total reward attached to the actions that the state enables. Note that in PEPA no reward can be attached to internal, τ, actions.

4.1 Case Studies

As originally intended, PEPA has been applied to study the performance characteristics of a number of computer and communication systems. Initial examples focussed on well-known standard performance evaluation abstractions such as *multi-server multi-queue* systems [41] and various queueing systems [4]. However over time more realistic case studies emerged, both from the PEPA group and from others. For example, in [42] the performance impact of fault-tolerant protocols within a distributed system framework is evaluated. In [5] Bowman *et al.* develop a model of multimedia traffic characteristics and use it to derive quality of service measures such as jitter, throughput and latency. In an investigation of ways in which to ease the development of parallel database systems, the STEADY group at the Heriot-Watt University proposed the use of *performance estimators*. PEPA was used to verify the output of the performance estimators for a number of particular configurations and therefore improve confidence in the approach [43].

In recent work a group at the PRiSM Laboratory of the University of Versailles are working on a novel active rule-based approach to active networks (networks in which intermediate nodes supplement routing of data with some computation) [44]. A PEPA model was used to study the impact of the "*active*" traffic on the non-active cross-traffic in terms of loss rate and latency within an active switch

[45]. Furthermore the models were validated against simulation models of the same system and showed very good agreement [46].

In addition, the formalism has been applied to a number of other problems which are beyond the usual arena of computer performance evaluation.

Inland shipping. Luk Knapen of Hasselt applied PEPA to study traffic flow within the inland shipping network of Belgium focussing in particular on the locks and movable bridges.

Robotic workcells. Robert Holton of the University of Bradford used PEPA models to analyse the performance and functional correctness of a robotic workcell designed for a automated manufacturing system [3,2].

Cellular telephone networks. A team from the PRiSM Laboratory at the University of Versailles considered a problem of dimensioning in a cellular telephone network. They used a PEPA model to study the impact on call blocking and dropping of allocating bandwidth resources between micro and macro-cell level [8]. They took advantage of automatic aggregation [47].

Automotive diagnostic expert systems. Console *et al.* of the University of Turin constructed a PEPA model of an automatic diagnostic system to be deployed in a car. A large number of sensors were placed around the car and some number could trigger an alarm. The role of the PEPA model was to provide probabilistic reasoning to resolve the likely cause of the alarm based on previous observations of the timing and frequency of individual faults [48].

5 Tool Support

Case studies of the size and complexity described above are only possible if the modelling process has adequate support. In this section we describe some of the tool support which is available for performance modelling using stochastic process algebras. We focus primarily on the tools which support PEPA and the analysis techniques that they offer. There is a brief discussion of other SPA tools at the end of the section.

5.1 PEPA Tools

The PEPA Plugin Project. The PEPA Plugin Project is a software tool for reasoning about the various stages of the Markovian analysis of PEPA models. The tool is implemented as a collection of plug-ins for Eclipse [49], an extensible integrated development environment for a large variety of programming and modelling languages such as Java, C++, Python and UML. This framework was chosen for three main reasons. First, Eclipse is a freely available product. Second, it is widely supported by a growing community of users and businesses. Third, it can run on a variety of platforms, as it is implemented in Java and the graphical library used for the user interface is available on many operating systems.

The functionalities of the tool are accessible both programmatically and through a more user-friendly graphical interface. In the remainder of this section we focus on the latter method. Resources of an Eclipse workspace can be

manipulated using two main classes of tools, *editors* and *views*. The former follow the traditional open-save-close cycle pattern. The latter are typically used to navigate resources, modify properties of a resource and provide additional information on the resource being edited.

The PEPA Plugin contributes an editor for the language and views which assist the user during the entire cycle of model development. Static analysis is used for checking the well-formedness of a model and detecting potential errors prior to inferring the derivation graph of the system. A well-formed model can be derived, i.e. the underlying Markov process is extracted and the corresponding state space can thus be navigated and filtered via the *State Space* view. Finally, the CTMC allows numerical steady-state analyses such as activity throughput and component utilisation.

Editor. A PEPA editor is opened for files in the Eclipse workspace which have the `pepa` extension. The editor provides a convenient way to run a parser which translates the model description in the PEPA language into an in-memory representation suitable for further processing. This form is represented graphically in the *AST* view by means of a hierarchical structure for the model. The in-memory model also acts as an intermediate form for converting PEPA models into external formats. In particular, the PEPA plugin project provides an exporter to EMF [50], the de-facto standard for data exchange within the Eclipse framework.

Static Analysis. Static analysis deals with checking the well-formedness of a PEPA model. Because of its low computational cost, static analysis is performed every time the text of the model description is saved. The output of this tool is a contribution of a list of messages to the already existing Eclipse *Problem* view. The information provided can be grouped into two categories: *warnings* are messages about low priority problems which do not prevent further processing; *errors* are instead critical problems which must be fixed in order to continue the model development process. For instance, basic warning messages are about rate or process definitions which are defined in the model description but never used; error messages can be about rate or process names which are used but never defined.[3]

More advanced static analysis is carried out to detect potential local deadlocks, redundant declaration of actions of the cooperation operator and unguarded component uses giving rise to non-well-founded definitions of processes, i.e. self-containing processes. In order to fulfill these tasks, the model's in-memory representation is iteratively walked to create two support data structures: *complete action types set* and *used definition set*.

The complete action type set $\mathcal{A}$ of a component is the set of all the action types which can performed by the component during its evolution. This set can be calculated according to Tab. 2. For example, if we consider the model in Fig. 5 (for the sake of clarity we omit the actual rate values) then the complete action type sets of its constants are as follows:

[3] It is worthwhile noting that rate names must be declared before using them in a prefix definition. However, there is no such a rule with regards to process definitions.

$$\begin{aligned}
\boldsymbol{\mathcal{A}}(P1) &= \{\alpha, \beta, \gamma, \delta\} \\
\boldsymbol{\mathcal{A}}(P2) &= \{\alpha, \beta, \gamma, \delta\} \\
\boldsymbol{\mathcal{A}}(P3) &= \{\alpha, \beta, \gamma, \delta\} \\
\boldsymbol{\mathcal{A}}(Q1) &= \{\alpha, \beta, \epsilon, \eta\} \\
\boldsymbol{\mathcal{A}}(Q2) &= \{\alpha, \beta, \epsilon, \eta\} \\
\boldsymbol{\mathcal{A}}(Q3) &= \{\alpha, \beta, \epsilon, \eta\}
\end{aligned} \tag{5.1}$$

The used definition set $\boldsymbol{\mathcal{U}}$ of a component is the set of all the constants which the

Table 2. Rules for deriving the complete action type set

Constant $A \stackrel{\text{def}}{=} P$	$\boldsymbol{\mathcal{A}}(A) = \boldsymbol{\mathcal{A}}(P)$
Prefix $(\alpha, r).P$	$\boldsymbol{\mathcal{A}}\big((\alpha, r).P\big) = \{\alpha\} \cup \boldsymbol{\mathcal{A}}(P)$
Choice $P + Q$	$\boldsymbol{\mathcal{A}}(P + Q) = \boldsymbol{\mathcal{A}}(P) \cup \boldsymbol{\mathcal{A}}(Q)$
Cooperation $P \underset{L}{\bowtie} Q$	$\boldsymbol{\mathcal{A}}(P \underset{L}{\bowtie} Q) = \boldsymbol{\mathcal{A}}(P) \cup \boldsymbol{\mathcal{A}}(Q)$
Hiding $P\backslash\{L\}$	$\boldsymbol{\mathcal{A}}(P\backslash\{L\}) = \boldsymbol{\mathcal{A}}(P) - L$

component behaves as during its evolution. This set can be calculated according to the rules in Tab. 3. The used definition sets of the constants of the model in Fig. 5 are as follows:

$$\begin{aligned}
\boldsymbol{\mathcal{U}}(P1) &= \{P1, P2, P3\} \\
\boldsymbol{\mathcal{U}}(P2) &= \{P1, P2, P3\} \\
\boldsymbol{\mathcal{U}}(P3) &= \{P1, P2, P3\} \\
\boldsymbol{\mathcal{U}}(Q1) &= \{Q1, Q2, Q3\} \\
\boldsymbol{\mathcal{U}}(Q2) &= \{Q1, Q2, Q3\} \\
\boldsymbol{\mathcal{U}}(Q3) &= \{Q1, Q2, Q3\}
\end{aligned} \tag{5.2}$$

$$\begin{aligned}
P1 &\stackrel{\text{def}}{=} (\alpha, r).P2 + (\beta, s).P3 \\
P2 &\stackrel{\text{def}}{=} (\gamma, t).P1 \\
P3 &\stackrel{\text{def}}{=} (\delta, u).P1 \\
Q1 &\stackrel{\text{def}}{=} (\alpha, \top).Q2 + (\beta, \top).Q3 \\
Q2 &\stackrel{\text{def}}{=} (\epsilon, v).Q1 \\
Q3 &\stackrel{\text{def}}{=} (\eta, w).Q1 \\
P1 &\underset{\{\alpha,\beta\}}{\bowtie} Q1
\end{aligned}$$

Fig. 5. An example of PEPA model

Let us consider two processes which cooperate over a non-empty set of action types. A local deadlock is a condition that may occur when one process cannot proceed because it is in a state where it is synchronised on an activity which can never be performed by its partner. The model in Fig. 6 exhibits a local deadlock

Table 3. Rules for deriving the used definition set

Constant $A \stackrel{\text{def}}{=} P$	$\{P\} \cup \boldsymbol{\mathcal{U}}(P)$
Prefix $(\alpha, r).P$	$\boldsymbol{\mathcal{U}}(P)$
Choice $P + Q$	$\boldsymbol{\mathcal{U}}(P) \cup \boldsymbol{\mathcal{U}}(Q)$
Cooperation $P \underset{L}{\bowtie} Q$	$\boldsymbol{\mathcal{U}}(Q) \cup \boldsymbol{\mathcal{U}}(R)$
Hiding $P \backslash \{L\}$	$\boldsymbol{\mathcal{U}}(P)$

in the initial state, because the action type α cannot be performed by either *Q1* or *Q2*. Local deadlock conditions are critical errors which can be statically detected by examining the used definition set of each cooperation of a model. In particular, a cooperation $P \underset{L}{\bowtie} Q$ gives rise to a deadlock on the action α if the following condition holds:

$$\exists \alpha \in L : \alpha \notin \boldsymbol{\mathcal{A}}(P) \cap \boldsymbol{\mathcal{A}}(Q), \alpha \in \boldsymbol{\mathcal{A}}(P) \cup \boldsymbol{\mathcal{A}}(Q) \tag{5.3}$$

The tool emits warning messages if it discovers the existence of redundant definition of action types in cooperation sets. A cooperation specifies a redundant action type α if the following condition holds:

$$\exists \alpha \in L : \alpha \notin \boldsymbol{\mathcal{A}}(P) \cup \boldsymbol{\mathcal{A}}(Q) \tag{5.4}$$

The used definition set allows for the detection of non-guarded recursive definitions of components. In Fig. 7 is shown a model exhibiting such a condition. A subset of the infinite labeled transition system of component *P1* is:

$$\begin{aligned} P1 &\xrightarrow{(\gamma,t)} P2 \parallel P3 \parallel P3 \\ &\xrightarrow{(\gamma,t)} P2 \parallel P3 \parallel P3 \parallel P3 \\ &\xrightarrow{(\gamma,t)} P2 \parallel P3 \parallel P3 \parallel P3 \parallel P3 \\ &\xrightarrow{(\gamma,t)} \cdots \end{aligned}$$

This gives rise to an infinite-state Markov chain. We wish to work with finite-state Markov chains so we reject definitions such as these as being ill-formed.

$$\begin{aligned} P1 &\stackrel{\text{def}}{=} (\alpha, r).P2 \\ P2 &\stackrel{\text{def}}{=} (\gamma, t).P1 \\ Q1 &\stackrel{\text{def}}{=} (\beta, s).Q2 \\ Q2 &\stackrel{\text{def}}{=} (\epsilon, v).Q1 \\ P1 &\underset{\{\alpha\}}{\bowtie} Q1 \end{aligned}$$

Fig. 6. Example of a PEPA model with local deadlock

$$\begin{array}{l} \ldots \\ P1 \stackrel{\text{def}}{=} P2 \parallel P3 \\ P2 \stackrel{\text{def}}{=} (\gamma, t).P1 \\ \ldots \end{array}$$

Fig. 7. Example of a PEPA model with non-guarded recursive definitions of components

The used definition set is calculated for each process constant $A \stackrel{\text{def}}{=} P$ which defines a cooperation (in the example, $\boldsymbol{\mathcal{U}}(P1)$ would be calculated). Such a constant is not well-formed if the following condition holds:

$$A \in \boldsymbol{\mathcal{U}}(A) \tag{5.5}$$

The PEPA Plugin project provides a tool for state space derivation, i.e. the process of extracting a Markov process from the labeled transition system of the PEPA model. The output of the tool is the state space and the corresponding infinitesimal generator of the CTMC. The state space can be navigated and filtered via the *State Space* view. The state space is represented in a tabular form: the first column is the state number; then follow as many columns as the number of top-level components of the system. The tabular representation of the state space of the model in Fig. 5 would be as in Tab. 4.

Table 4. Tabular representation of the state space of the example model

State Number	First Component	Second Component
1	*P1*	*Q1*
2	*P3*	*Q3*
3	*P3*	*Q1*
4	*P1*	*Q3*
5	*P2*	*Q2*
6	*P2*	*Q1*
7	*P1*	*Q2*

A variety of filter options is available in order to narrow down the number of states shown in the view. The user can exclude/include states which have a sequential component in a particular local state or states which contain unnamed processes (i.e., prefixes). More precise filtering can be obtained by means of a pattern language which allows the user to match local states which contain given top-level component local states at specified positions. The components are separated by vertical bars and a wild-card is used to disregard positions which are of no interest. According to the example in Fig. 5, the pattern `P1 | *` would match the states whose first top-level component is in state P1,

thus displaying states 1,4,7; the pattern `* | Q2` would match states 5,7. For a more concise description of the filter, the generic pattern `P` is considered as an abbreviation of `P | * | ... | *`.

Additionally, the plug-in contributes the *Single Step Navigator*, a tool for walking the state space. This is particularly useful for debugging purposes. It consists of two tables containing the list of incoming and outgoing states. The sequential components which cause the transition to be performed are highlighted and an option allows the user to make filtered states not walkable.

A model whose state space is derived successfully is amenable to performance analysis which can be carried out by calculating the steady-state probability distribution of the CTMC over the state space. The user interface provides a dialogue wizard which guides the user through this process. The wizard is a graphical interface to the MTJ toolkit [51], the library used for the numerical solution of the Markov chain, allowing the user to choose and tune the parameters of an extensive selection of solvers and preconditioners.

After the model is successfully solved, the State Space view is updated with information on the obtained steady-state probability distribution which is shown on an additional column. Additional analysis can be carried out via the *Performance Evaluation* view, which permits throughput and utilisation analysis. Throughput is an action-related metric showing the rate at which an action is performed at steady-state; utilisation is related to a sequential component showing the steady-state distribution probability over its local states.

In order to better illustrate these metrics, let us consider the model in Fig. 8 consisting of one single sequential component evolving through three local states. With rates $r = 2, s = 1, t = 1$ the utilisation figures are $P1 = 0.2, P2 = 0.4, P3 = 0.4$ whereas throughput is 0.4 for each action.

$$
\begin{aligned}
P1 &\stackrel{\text{def}}{=} (\alpha, r).P2 \\
P2 &\stackrel{\text{def}}{=} (\beta, s).P3 \\
P3 &\stackrel{\text{def}}{=} (\gamma, t).P1
\end{aligned}
$$

Fig. 8. A tiny PEPA model with one sequential component

Experimentation is a tool for sensitivity analysis. The user can supply ranges for rate values against which the performance metrics described above are calculated. Results are then shown in the form of graphs for which a number of exporting options are available.

The Imperial PEPA Compiler. The Imperial PEPA Compiler (IPC) [52] provides an alternative implementation of the PEPA language, providing a bridge to performance analysis tools developed at Imperial College by Knottenbelt and his group [53,54].

The ipc tool translates an input PEPA model into the Petri net notation provided by Dnamaca [53]. Its support for the PEPA language is comprehensive.

Apparent rates are supported, as are anonymous components. The great advantage of accessing the functionality of the Dnamaca analyser is that other forms of analysis (beyond steady-state) become available.

The steady state probability distribution represents the behaviour of the system at equilibrium, where the influence of the initial state of the system is no longer measurable. Some performance measures of interest cannot be derived from the results of steady state analysis. Examples of performance measures in the class of non-equilibrium measurements include *mean time to failure* analysis, as computed in the evaluation of dependable systems. Other examples include the probabilistic quality-of-service guarantees which underpin most commercial service level agreements (SLAs): e.g. the probability that a 10-node cluster should be able to process 3000 database transactions in less than 6 seconds should be greater than 0.915; or a train service should not run more than 10 minutes late more than 20% of the time.

More generally, such measures necessitate the computation of *passage-time quantiles* which detail the probability of passing through the system evolution from a start state to an end state (or a set of starting states to a set of end states). The computation of such measures depends on the aggregate time behaviour across a whole system of complex interactions. The computation of passage-time quantiles depends on *transient analysis* of the CTMC, which is more expensive than steady-state analysis in both run-time and memory consumption.

Via ipc, the unique solution capabilities of Dnamaca become available and because of this it is possible to efficiently perform passage time analysis over PEPA models [52,54]. Start and end points are specified using the concept of *stochastic probes* developed by Argent-Katwala, Bradley and Dingle [55]. Stochastic probes are themselves PEPA components which have been generated from regular expression-based inputs.

The PRISM model checker. PRISM is a probabilistic model checker developed by Kwiatkowska's group at the University of Birmingham. It supports discrete time Markov chains and Markov decision processes as well as CTMCs. The standard input to PRISM is a model described in a simple reactive modules language. PEPA was integrated into the tool via a compiler which translates PEPA models into this language. The developers at the University of Birmingham extended PRISM's modelling capabilities to implement at the binary decision diagram level PEPA's combinators (cooperation and hiding).

Integration into PRISM enables model checking of the CTMC underlying a PEPA model against properties expressed in Continuous Stochastic Logic (CSL) [56]. It also provides access to the efficient numerical solutions of PRISM based on MTBDDs [57] and sparse matrix representation. PRISM has been applied successfully to a number of PEPA (and PEPA net) case studies [58,59].

The Möbius modelling platform. The Möbius modelling framework [60] was developed at the University of Illinois Urbana-Champaign. It is both a multi-formalism and multi-paradigm modelling tool, i.e. it aims to offer the user a choice of model description techniques and solution methods. Moreover it is

designed to allow a model to be composed of submodels which may be expressed in different formalisms. It has a broad spectrum of users in North America. Integrating PEPA into Möbius offered opportunities to present stochastic process algebra to users who were previously unfamiliar with the formalism, and to expore the possibilities of interaction between modelling formalisms [61].

5.2 Related Work

Over the years, several software tools have been made available for supporting computer-aided analysis with process algebra. TwoTowers [62], for example, provides a similar range of tools for the stochastic process algebra EMPAgr.

TIPP-Tool. The TIPP-Tool is a prototype modelling tool for creating and evaluation TIPP models of parallel and distributed systems. It supports a LOTOS-oriented input language and as well as facilities to apply functional analysis based on reachability analysis, it provides a set of numerical solution modules for the stationary and transient analysis of the Markov process underlying a TIPP specification [63,64].

In order to evaluate the performance of the specification the derived transition system serves as a base for further reduction into a Markov process. For the steady state analysis of the underlying Markov process a variety of numerical algorithms are available: LU-factorization, power method, and Gauss-Seidel iteration scheme. TIPP-Tool supports also transient analysis by providing methods to compute the mean time to absorption of an absorbing Markov chain or the transient state probabilities. For the latter a refined randomisation scheme is provided.

The result of numerical analysis is usually a vector with state probabilities. In order to obtain more sophisticated and expressive results the user can specify measures. This is done via rewards that are assigned to states which match a regular expression which the user must specify. Series of experiments are also supported by allowing rates to be symbolic variables. The specification of the model, as well as the measures and experiments, is supported through a graphical user interface.

TwoTowers. TwoTowers [62], which supports modelling with the SPA language EMPA, builds on two existing tools, CWB-NC [65] for functional analysis and MARCA [66] for performance analysis. The specification language for TwoTowers is EMPA$_r$, an extension of EMPA to include the specification of rewards—in the subsequent analysis these rewards are used to derive performance measures. The other SPA tools include the facility to associate a reward structure with a model; in EMPA$_r$ the reward structure is assumed to be an integral part of the model.

TwoTowers has a graphical user interface written in Tcl/Tk. This interface allows the user to edit and compile specifications and provides access to the various analysis routines. CWB-NC provides a suite of functional analysis techniques: model checking, equivalence checking, preorder checking and reachability analysis. MARCA provides for both steady state and transient performance analysis of the underlying Markov process. In addition there is a simulation engine which allows models to be simulated.

More information about TwoTowers is available at:
(`http://www.sti.uniurb.it/bernardo/twotowers`).

MoDeST. The MoDeST modelling language (Modelling and Description language for Stochastic Timed systems) [33] enriches a process algebra with atomic statements to control the granularity of transitions, non-deterministic and probabilistic branching and timing. The MoDeST language provides conventional programming constructs such as iteration, alternatives, atomic statements, and exception handling in the style of user-friendly specification languages such as Promela. The MoDeST semantics maps each MoDeST specification onto a *stochastic timed automata*, a modelling formalism which subsumes timed, stochastic and probabilistic automata.

The MoDeST language has been integrated into the Möbius multi-paradigm modelling framework [67] as an atomic model. MoDeST models which do not use non-determinism can be assessed quantitatively using the discrete-event simulator of Möbius or its Markovian analysers. MoDeST models are mapped onto C++ code which links against the Möbius Abstract Functional Interface (AFI).

Verification of properties of MoDeST models can be performed using the CADP Tools [68].

6 Case Studies

6.1 Roland the Gunslinger

In this subsection we consider a sequence of small examples based around a character known as "Roland the Gunslinger". These simple models are intended to be intuitive to understand but yet show some of the main features of the language and demonstrate a variety of solution techniques. A more substantial example is presented in the following subsection.

Roland alone. Roland Deschain is a gunslinger and his primary activity is firing his gun which is a six-shooter, i.e. there is room in the barrel for six bullets at a time. When his gun is empty Roland will reload the gun and then continue shooting.

$$
\begin{aligned}
Roland_6 &\stackrel{\text{def}}{=} (fire, r_{fire}).Roland_5 \\
Roland_5 &\stackrel{\text{def}}{=} (fire, r_{fire}).Roland_4 \\
Roland_4 &\stackrel{\text{def}}{=} (fire, r_{fire}).Roland_3 \\
Roland_3 &\stackrel{\text{def}}{=} (fire, r_{fire}).Roland_2 \\
Roland_2 &\stackrel{\text{def}}{=} (fire, r_{fire}).Roland_1 \\
Roland_1 &\stackrel{\text{def}}{=} (fire, r_{fire}).Roland_{empty} \\
Roland_{empty} &\stackrel{\text{def}}{=} (reload, r_{reload}).Roland_6
\end{aligned}
$$

If we suppose that Roland has two guns then he should be allowed to fire either gun independently. A simplistic way to model this is to have two instances of *Roland* in parallel:

$$Roland_6 \parallel Roland_6$$

However, this model does not capture the fact that Roland needs both hands in order to reload either gun. The simplest way to fix this is to assume that Roland only reloads both guns when both are empty.

$$Roland_6 \underset{\{reload\}}{\bowtie} Roland_6$$

In the remaining models, we consider only the case of Roland using his shotgun, which has only two bullets before it needs reloading, and requires both hands for firing.

Choice. In the first straightforward model of Roland, he was simply firing his guns. We now consider a model which captures the possibility that Roland will miss or hit his target.

Upon his travels Roland will encounter some enemies with whom he will have no choice but to enter combat. In this model it is assumed that his enemies do not possess the skill required to seriously harm Roland. Therefore he never dies but simply encounters villians and fires at them until he successfully hits them. Each hit is assumed to be fatal and it is assumed that a sense of honour prevents an enemy from attacking Roland if he is already involved in a gun fight.

The rates involved in this model are given in Table 5; each is measured in seconds, so a rate of 1.0 would indicate that the action is expected to occur once every second. There is one special parameter, $p_{hit\text{-}success}$ which is a probability measure, used to calculate the values for the rates r_{hit} and r_{miss}.

$$\begin{aligned}
Roland_{idle} &\stackrel{\text{def}}{=} (attack, r_{attack}).Roland_2 \\
Roland_2 &\stackrel{\text{def}}{=} (hit, r_{hit}).(reload, r_{reload}).Roland_{idle} + (miss, r_{miss}).Roland_1 \\
Roland_1 &\stackrel{\text{def}}{=} (hit, r_{hit}).(reload, r_{reload}).Roland_{idle} + (miss, r_{miss}).Roland_{empty} \\
Roland_{empty} &\stackrel{\text{def}}{=} (reload, r_{reload}).Roland_2
\end{aligned}$$

Table 5. Parameter settings for the $Roland_2$ model

parameter	*value*	*explanation*
r_{fire}	1.0	Roland can fire the gun once per-second
$p_{hit\text{-}success}$	0.8	Roland has an 80% success rate
r_{hit}	0.8	$r_{fire} \times p_{hit\text{-}success}$
r_{miss}	0.2	$r_{fire} \times (1 - p_{hit\text{-}success})$
r_{reload}	0.3	It takes Roland about 3 seconds to reload
r_{attack}	0.01	Roland is attacked once every 100 seconds

Steady-State Analysis. This model can be used to calculate the probability that at any time Roland is involved in a battle. Using steady state analysis this amounts to calculating the probability that *Roland* is in any of the states in which a battle is on-going, i.e. $Roland_2$, $Roland_1$ and $Roland_{empty}$. Alternatively one can calculate the probability that *Roland* is in the single peaceful state $Roland_{idle}$ and subtract it from 1. This was done for the above model, for the parameter values shown in Table 5, giving the result:

```
State Measure 'roland peaceful' % 100 seconds

 mean          9.5490716180e-01
```

This shows that there is more than a 95 percent chance that Roland is not currently involved in a gun battle. This is intuitively what we would expect since he is attacked once every 100 seconds and will usually take around one second to fire each bullet. Two bullets then cost him a further three seconds to reload, however since his success rate is at 80 percent, he will often not need to reload.

Transient Analysis. Transient analysis could be used here to determine the probability that Roland will have killed some enemy within a given time, say two minutes, of starting off on his travels.

Passage-Time Analysis. An example of a passage-time analysis for this model would calculate the probability that at a given time after he is attacked, Roland has killed his attacker. This would involve calculating the probability that the model performs a *hit* action within the given time after performing an *attack* action.

The graph on the left hand side of Figure 9 shows an example of this kind of analysis. It shows the probability that Roland will successfully perform a *hit* action a given time after an *attack* action. This also confirms our instincts concerning the steady-state analysis. Since there is a 95 percent chance that Roland is not involved in a gun battle, and one occurs about once every 100 seconds, then we should expect gun battles to last for around five seconds. Looking at the graph on the left hand side of Figure 9 we see that the probability that Roland has performed a *hit* action five seconds after an *attack* action is quite high at just over 90 percent.

We can also measure, for example, the probability that Roland will miss after having been attacked. This probability is somewhat low. One of the reasons is that, if Roland hits the target with his first shot then in order to observe a miss action in the model we will have to wait until Roland is attacked again. The graph on the right hand side of Figure 9 shows the probability curve for the same time period as the first graph. Because the attack rate is low, in this period of time it is unlikely that Roland will be attacked for a second time. For this reason the graph looks similar to the first graph, but translated down the probability axis. The initial rise in probability corresponds, as in the first graph, to the probability that Roland will fire his gun within that time.

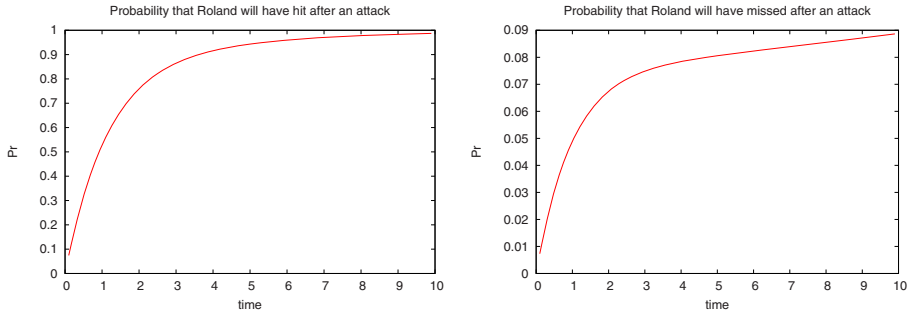


Fig. 9. Probability of events occurring after an attack event

Cooperation. We now consider the model augmented to allow the enemies of Roland to fight back. However, they are currently somewhat ineffective and always miss Roland when they fire. (The next model will fix this.) This model can be used to calculate properties such as the likelihood that an enemy will manage to fire one shot before they are killed by Roland.

Table 6. Parameters for the enemies

parameter	*value*	*explanation*
r_{attack}	0.01	Roland is attacked once every 100 seconds
$r_{e\text{-}miss}$	0.3	Enemies can fire only once every 3 seconds

$$
\begin{aligned}
Roland_{idle} &\stackrel{\text{def}}{=} (attack, \top).Roland_2 \\
Roland_2 &\stackrel{\text{def}}{=} (hit, r_{hit}).(reload, r_{reload}).Roland_{idle} + (miss, r_{miss}).Roland_1 \\
Roland_1 &\stackrel{\text{def}}{=} (hit, r_{hit}).(reload, r_{reload}).Roland_{idle} + (miss, r_{miss}).Roland_{empty} \\
Roland_{empty} &\stackrel{\text{def}}{=} (reload, r_{reload}).Roland_2
\end{aligned}
$$

$$
\begin{aligned}
Enemies_{idle} &\stackrel{\text{def}}{=} (attack, r_{attack}).Enemies_{attack} \\
Enemies_{attack} &\stackrel{\text{def}}{=} (fire, r_{e\text{-}miss}).Enemies_{attack} + (hit, \top).Enemies_{idle}
\end{aligned}
$$

$$
Roland_2 \underset{\{hit\}}{\bowtie} Enemies_{idle}
$$

Notice that in this model the behaviour of the enemy has been simplified. There is no running out of bullets or reloading. This model can be thought of as an approximation to a more complicated component similar to the one which models Roland. The rate at which the enemy fires encompasses all of the actions, including the reloading of an empty gun. The analyses associated with this model are very similar to those for the previous model. Steady-state analysis can be used to determine the likelihood that Roland is currently in a peaceful state.

It is also sometimes useful to carry out a validation of the model by calculating a metric which we believe we already know the value of. For example in this model we could make such a sanity check by calculating the probability that the model is in a state in which Roland is idle but the enemies are not, or vice versa. This should never occur and hence the probability should be zero.

Transient analysis could again be used to calculate the expected time before Roland is attacked or the expected time before Roland has made a kill.

Sensitivity Analysis. Due to the roles which activities play in creating the dynamics of our stochastic process algebra model it may be that increasing the rate of one activity increases the score obtained by the model on our chosen performance measure of interest. Conversely, increasing the rate of another activity may decrease the score which we get. Changing one rate a little may vary the score a lot. Changing another rate a lot might only vary the score a little. The study of how changes in performance depend on changes in parameter values in this way is known as *sensitivity analysis*.

Sensitivity analysis is performed by solving the model many times while varying the rates slightly. For this model we chose to vary three of the rates involved and measured for each combination of rates the passage-time probability that the model performs a *hit* action after performing an *attack* action.

The results are shown in Figure 10. The first three graphs measure the sensitivity of each of the three rates. The top left graph shows the effect that varying the $p_{hit\text{-}success}$ parameter has. The top right graph depicts the effect of varying the r_{fire} rate and finally the middle left graph shows the r_{reload} rate.

From these graphs one can deduce that the strongest influence is from the $p_{hit\text{-}success}$ parameter. To see this, notice the greater curvature of the graph of probability against time as the value of the $p_{hit\text{-}success}$ parameter is increased. In contrast, the top right and middle left graphs show little of the warping that is seen in the first graph.

In the final three graphs we measure the effect that varying one rate has, on the effect of another rate. In most models the effect which one rate has depends on the values of the other rates. For example, in our model, clearly if both the r_{reload} and r_{fire} rates are small then the effect of the $p_{hit\text{-}success}$ is large since Roland pays a large penalty whenever he misses. If, however, these two rates are large then Roland pays less of a penalty for missing and hence the effect of increasing (or decreasing) $p_{hit\text{-}success}$ is diluted.

To keep such graphs comprehensible to humans we fix the time at which the probability is measured. The first of these graphs on the middle right of Figure 10 measures the effect of varying r_{reload} against $p_{hit\text{-}success}$. Similarly the bottom left graph depicts varying r_{reload} against r_{fire}; and finally, the bottom left varies r_{fire} and $p_{hit\text{-}success}$. This final graph is interesting. On the far left it can be seen that when the r_{fire} is low, as we increase $p_{hit\text{-}success}$ there is close to a linear increase in the probability. However, when the r_{fire} is high the graph of probability against $p_{hit\text{-}success}$ rises sharply and then becomes less steep. This is most likely because when the r_{fire} rate is high, the penalty for Roland reloading is also relatively high in comparison and therefore the benefit of avoiding this is greater.

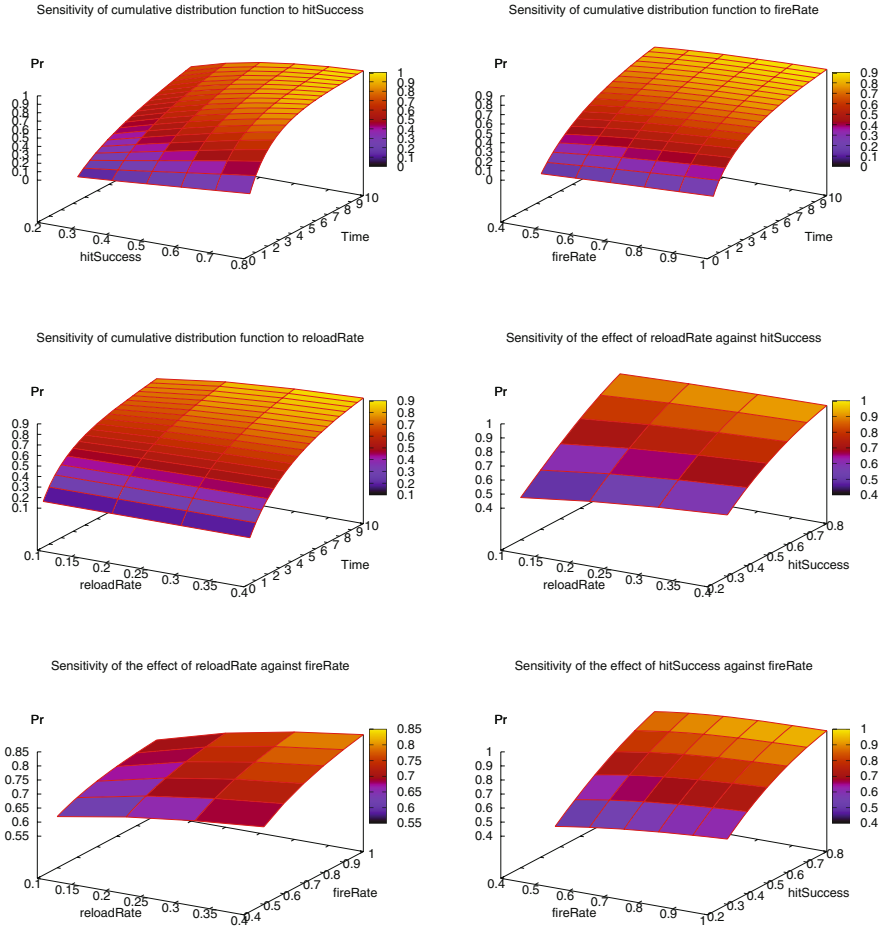


Fig. 10. Graphs of cumulative distribution function sensitivity to changes in rates for the passage from attack to Roland killing the enemy

Accurate Enemies. We now allow the enemies of Roland to actually hit him. This means that Roland may die. It is important to note that this has the consequence that the model will always deadlock. The underlying Markov process is no longer ergodic.

To maintain the simplicity of the model we assume that the enemies can only hit Roland once every 50 seconds. Note that this rate approximates the rate of a more detailed model in which we would assign a process to the enemies which is much like that of the process which describes Roland. That is, it can fire and miss, run out of bullets and reload etc. before finally hitting Roland. The only new parameter is $r_{e\text{-}hit}$ which is assigned a value 0.02 to reflect this assumption.

$$
\begin{aligned}
Roland_{idle} &\stackrel{\text{def}}{=} (attack, \top).Roland_2 \\
Roland_2 &\stackrel{\text{def}}{=} (hit, r_{hit}).(reload, r_{reload}).Roland_{idle} + (miss, r_{miss}).Roland_1 \\
&\quad + (e\text{-}hit, \top).Roland_{dead} \\
Roland_1 &\stackrel{\text{def}}{=} (hit, r_{hit}).(reload, r_{reload}).Roland_{idle} + (miss, r_{miss}).Roland_{empty} \\
&\quad + (e\text{-}hit, \top).Roland_{dead} \\
Roland_{empty} &\stackrel{\text{def}}{=} (reload, r_{reload}).(reload, r_{reload}).Roland_2 + (e\text{-}hit, \top).Roland_{dead} \\
Roland_{dead} &\stackrel{\text{def}}{=} Stop
\end{aligned}
$$

$$
\begin{aligned}
Enemies_{idle} &\stackrel{\text{def}}{=} (attack, r_{attack}).Enemies_{attack} \\
Enemies_{attack} &\stackrel{\text{def}}{=} (e\text{-}hit, r_{e\text{-}hit}).Enemies_{idle} + (hit, \top).Enemies_{idle}
\end{aligned}
$$

$$
Roland_2 \underset{\{hit, attack, e\text{-}hit\}}{\bowtie} Enemies_{idle}
$$

Steady-State Analysis. This model has the interesting property that the model will always deadlock: because there is an infinite supply of enemies eventually Roland will always die. This means that steady-state analysis would not be used on such a model, although a possible use would be as a validation of the model, as was done for the previous model.

Transient Analysis. Transient analysis on this model can be used to calculate the expected time at which Roland will die, or rather the probability that Roland is dead after a given amount of time. As the time increases this should tend towards probability 1.

Passage-Time Analysis. As before, the passage-time analysis on this model would be used to calculate the probability of a given event happening at a given time after another given event. Here we might again choose the starting event to be an attack on Roland, and the ending event could be either Roland dying or Roland winning the gun fight.

More Cooperation. The cooperation so far has involved the synchronisation between two processes on events that they have either caused directly or are directly affected by. In this section cooperation is used to synchronise between components of the model such that they observe events which they neither directly cause nor are directly affected by. In this particular example an accomplice is befriended by Roland from time to time and whenever an enemy attacks, Roland and the accomplice fight together. Whenever either one of them kills the enemy the other must observe this action, so as to stop firing at a dead opponent.

The component representing Roland is now modified to include actions, which Roland does not participate in, such as his accomplice killing the enemy, but which nevertheless alter Roland's state and therefore must be witnessed.

Table 7. Parameter values for the accomplice

parameter	*value*	*explanation*
$r_{befriend}$	0.001	Roland befriends a stranger once every 1000 seconds
$r_{a\text{-}fire}$	1.0	the accomplice can also fire once per second
$p_{a\text{-}hit\text{-}success}$	0.6	the accomplice has a 60 percent accuracy
$r_{a\text{-}hit}$	0.6	$r_{fire} \times p_{hit\text{-}success}$
$r_{a\text{-}miss}$	0.4	$r_{fire} \times (1.0 - p_{hit\text{-}success})$
$r_{a\text{-}reload}$	0.25	it takes the accomplice 4 seconds to reload

$$
\begin{aligned}
Roland_{idle} &\stackrel{\text{def}}{=} (attack, \top).Roland_2 + (befriend, r_{befriend}).Roland_{idle} \\
Roland_2 &\stackrel{\text{def}}{=} (hit, r_{hit}).Roland_{hit} + (a\text{-}hit, \top).Roland_{idle} + (miss, r_{miss}).Roland_1 \\
Roland_1 &\stackrel{\text{def}}{=} (hit, r_{hit}).Roland_{hit} + (a\text{-}hit, \top).(reload, r_{reload}).Roland_{idle} \\
&\quad + (miss, r_{miss}).Roland_{empty} \\
Roland_{hit} &\stackrel{\text{def}}{=} (enemy\text{-}die, \top).(reload, r_{reload}).Roland_{idle} \\
Roland_{empty} &\stackrel{\text{def}}{=} (reload, r_{reload}).Roland_2 + (a\text{-}hit, \top).(reload, r_{reload}).Roland_{idle}
\end{aligned}
$$

The attack is assumed to be a concerted effort between Roland and his accomplice but we do not wish to leave Roland vulnerable when he has no accomplice. For this reason the representation of the accomplice includes a state when the accomplice is absent. In this state the accomplice component will passively participate in any attack which Roland makes. The alternative would be that Roland was blocked from attacking when he had no accomplice. Also note that, just as Roland witnesses if the accomplice kills the enemy, the accomplice also witnesses if Roland kills the enemy.

$$
\begin{aligned}
Accomplice_{abs} &\stackrel{\text{def}}{=} (befriend, r_{befriend}).Accomplice_{idle} + (hit, \top).Accomplice_{abs} \\
&\quad + (attack, \top).Accomplice_{abs}
\end{aligned}
$$

$$
\begin{aligned}
Accomplice_{idle} &\stackrel{\text{def}}{=} (attack, \top).Accomplice_2 \\
Accomplice_2 &\stackrel{\text{def}}{=} (a\text{-}hit, r_{a\text{-}hit}).Accomplice_{hit} + (hit, \top).Accomplice_{idle} \\
&\quad + (miss, r_{miss}).Accomplice_1 + (enemy\text{-}hit, \top).Accomplice_{abs} \\
Accomplice_1 &\stackrel{\text{def}}{=} (a\text{-}hit, r_{a\text{-}hit}).Accomplice_{hit} \\
&\quad + (hit, \top).(reload, r_{a\text{-}reload}).Accomplice_{idle} \\
&\quad + (miss, r_{miss}).Accomplice_{empty} + (enemy\text{-}hit, \top).Accomplice_{abs} \\
Accomplice_{hit} &\stackrel{\text{def}}{=} (enemy\text{-}die, \top).(reload, r_{a\text{-}reload}).Accomplice_{idle} \\
Accomplice_{empty} &\stackrel{\text{def}}{=} (reload, r_{a\text{-}reload}).Accomplice_2 + (enemy\text{-}hit, \top).Accomplice_{abs} \\
&\quad + (hit, \top).(reload, r_{a\text{-}reload}).Accomplice_{idle}
\end{aligned}
$$

The component representing the enemy is similar to before.

$$Enemies_{idle} \stackrel{\text{def}}{=} (attack, r_{attack}).Enemies_{attack}$$
$$Enemies_{attack} \stackrel{\text{def}}{=} (enemy\text{-}hit, r_{e\text{-}hit}).Enemies_{attack} + (enemy\text{-}die, \top).Enemies_{idle}$$

The system equation is as follows:

$$(Roland_2 \underset{\{hit, a\text{-}hit, befriend\}}{\bowtie} Accomplice_{abs}) \underset{\{attack, enemy\text{-}die, enemy\text{-}hit\}}{\bowtie} Enemies_{idle}$$

Steady-State Analysis. As before steady-state analysis can be used to determine the probability that at any given time Roland is involved in a gun battle. Additionally this can now be used to determine the likelihood that Roland is on his own or has an accomplice. It is interesting to note the relations between the rates involved in the model and the subsequent probabilities. Additionally the relations between each of the steady-state proabilities. Since Roland cannot perform a befriending action while currently involved in a confrontation with an enemy, the probabilty that Roland is in such a battle clearly affects the probability that he is alone in his quest. So for example if Roland's success rate is reduced then gun battles will take longer to resolve, hence Roland will be involved in a gun battle more often, and therefore he will befriend fewer accomplices.

Transient Analysis. An additional transient analysis would be to determine the expected time after Roland has set off before he meets his first accomplice.

Passage-Time Analysis. As with all the previous models the passage-time analysis will measure the probability starting from an *attack* action. Possible actions to stop the analysis at would be the event of the enemy's death or that of the accomplice. Since all gun battles now end in the enemy being killed stopping the analysis there would give us the expected duration of any one gun battle. Stopping the analysis with the death of the accomplice would also incorporate the chance that the enemy is killed but a further enemy attacks and hits the accomplice. However, the extra probability of this is rather small because of the low-rate at which Roland is attacked. Finally the passage-time analysis could be stopped on either of the two hit events, this would give us the probability at a given time after an attack event that either the accomplice or the enemy has been shot.

This model also presents a further possible starting action besides that of the *attack* action, that is the *befriend* action. An interesting passage-time query would be the probability that a given length of time after a *befriend* action has occurred that a *enemy-hit* action occurs. This would give the modeller an estimate of the duration of Roland's friendships.

Hiding. There is a possible deficiency in the above model. What if the enemy starts to perform a *befriend* action (or equally a *hit* or *a-hit* action)? This would invalidate our model as it would model strange things happening, for example if Roland would not be able to meet any new accomplices. Of course the problem is not that the enemy might use this as an underhand tactic since it is the modeller that is describing the enemy component. The problem is that the modeller

is fallible and may make a mistake, especially if the enemy component becomes more complex. One way to avoid this is to 'hide' those actions only Roland and the accomplice should cooperate on.

To do this for our model we can simply change the system equation to read:

$$((Roland_2 \underset{L_1}{\bowtie} Accomplice)/L_1) \underset{L_2}{\bowtie} Enemies_{idle}$$

where $L_1 = \{hit, a\text{-}hit, befriend\}$ and $L_2 = \{attack, enemy\text{-}die, enemy\text{-}hit\}$.

6.2 Web Service Composition

As example of a realistic case study, we consider an example of a business application which is composed from a number of offered web services. Furthermore there is an access control issue, as it must be ensured that the web service consumer has the requisite authority to execute the web services it requests. A schematic representation of the system is depicted in Fig. 11.

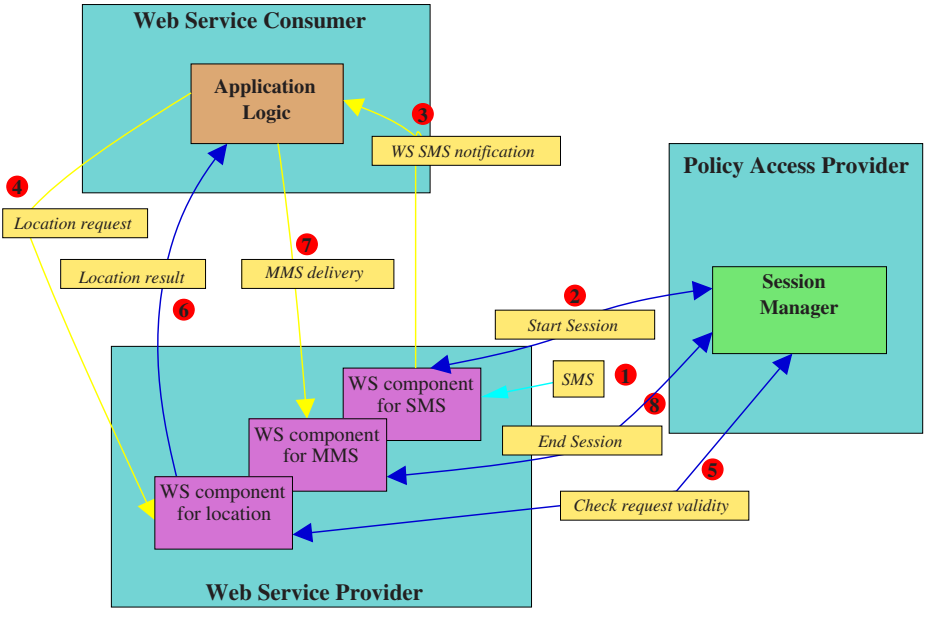


Fig. 11. Schematic representation of the web service composition

The scenario is as follows. Several web services are combined to define the business logic of an application. For example, consider an application to find the nearest restaurant for a user and show it on a map. This could involve web services for SMS and MMS handling in addition to the User Location web service. Moreover, a user should not be able to gain access to location information of an arbitary user.

This is where the access control aspect becomes important. Therefore, in addition to the requested web services, the web service provider may need to interact with some authorisation component to check that the current user has the correct authority to access the requested information. In adddition the service provider may stipulate some further conditions, such as that only one location request may be made per session:

1. The user activates a service by sending an SMS to a service centre number. This is handled by an appropriate web service.
2. This initiates a *start-session* message to be sent to the Policy Access Provider.
3. A notification is sent to the application that an SMS has arrived.
4. The application requests the user's location from a location web service.
5. The web service contacts the session manager within the policy access provider to check the validity of the request.
6. If the validity check is OK the location web service will return the location to the application which uses it to construct the appropriate map for the user.
7. This is then passed as an MMS to the MMS web service which delivers it to the user.
8. The MMS web service terminates the session with the Session Manager.

We model such a system with the following PEPA model. It has three types of model component, corresponding to the three large rectangles in Figure 11. Note that although the Web Service Provider consists of three distinct elements, we are interested in the session associated with each Web Service Consumer. Each session is associated with an instance of the Web Service Provider. Thus, concurrency is introduced into the model by allowing multiple sessions rather than by representing the constituent web services separately.

Component *Customer*. The customer's behaviour is simply modelled with two local states. In the first state the customer sends a request to the system via the *getSMS* action. She then waits for a response which triggers the *getMap* transition if it is successful. Therefore we associate the user-perceived system performance with the throughput of this action, which can be calculated directly from the steady-state probability distribution of the underlying Markov chain.

$$Customer \stackrel{\text{def}}{=} (getSMS, r_1).Customer_1$$
$$Customer_1 \stackrel{\text{def}}{=} (getMap, \top).Customer + (get404, \top).Customer$$

In this model sending either an error message *get404* or the requested map occur at the same rate *r8* and MMS passing between web services is ten times as fast as the communication with the user.

Component *WSConsumer*. The web service consumer, *WSConsumer*, follows a simple pattern of behaviour. Once it is notified that a session has been started by the user (via SMS message), it initiates a request for the user's current location and waits for a response. If the request was valid, the location is returned and used to compute the appropriate map for the user, which is then sent via an MMS message, using the web service for this.

$$
\begin{aligned}
WSConsumer &\stackrel{\text{def}}{=} (notify, \top).WSConsumer_2 \\
WSConsumer_2 &\stackrel{\text{def}}{=} (locReq, r_4).WSConsumer_3 \\
WSConsumer_3 &\stackrel{\text{def}}{=} (locRes, \top).WSConsumer_4 \\
&\quad + (locErr, \top).WSConsumer \\
WSConsumer_4 &\stackrel{\text{def}}{=} (compute, r_7).WSConsumer_5 \\
WSConsumer_5 &\stackrel{\text{def}}{=} (sendMMS, r_9).WSConsumer
\end{aligned}
$$

Component *WSProvider*. As explained above, although the Web Service Provider can be viewed as consisting of three independent web services, the use of sessions restricts a user's access to these services to be sequential. We assume that there is a distinct instance of the component *WSProvider* for each distinct session. As each would be in a distinct thread it is reasonable for there to be concurrency at this level. The activities of the component are as outlined in the scenario above. Note that the *checkValid* action is represented twice, to capture the two possible distinct outcomes of the action. If the check is successful the location must be returned to the Web Service Consumer in the form of a map (*getMap*). However, if the check revealed an invalid request (*locErr*) then an error must be returned to the Web Service Consumer (*get404*) and the session terminated (*stopSession*).

$$
\begin{aligned}
WSProvider &\stackrel{\text{def}}{=} (getSMS, \top).WSProvider_2 \\
WSProvider_2 &\stackrel{\text{def}}{=} (startSession, r_2).WSProvider_3 \\
WSProvider_3 &\stackrel{\text{def}}{=} (notify, r_3).WSProvider_4 \\
WSProvider_4 &\stackrel{\text{def}}{=} (locReq, \top).WSProvider_5 \\
WSProvider_5 &\stackrel{\text{def}}{=} (checkValid, 99 \cdot \top).WSProvider_6 \\
&\quad + (checkValid, \top).WSProvider_{10} \\
WSProvider_6 &\stackrel{\text{def}}{=} (locRes, r_6).WSProvider_7 \\
WSProvider_7 &\stackrel{\text{def}}{=} (sendMMS, \top).WSProvider_8 \\
WSProvider_8 &\stackrel{\text{def}}{=} (getMap, r_8).WSProvider_9 \\
WSProvider_9 &\stackrel{\text{def}}{=} (stopSession, r_2).WSProvider \\
WSProvider_{10} &\stackrel{\text{def}}{=} (locErr, r_6).WSProvider_{11} \\
WSProvider_{11} &\stackrel{\text{def}}{=} (get404, r_8).WSProvider_9
\end{aligned}
$$

Component *PAProvider*. In our model the Policy Access Provider has a very simple behaviour. It simply maintains a thread for each session and carries out the validity check on behalf of the Web Service Provider. This representation of the *PAProvider* is stateful.

$$
\begin{aligned}
PAProvider &\stackrel{\text{def}}{=} (startSession, \top).PAProvider_2 \\
PAProvider_2 &\stackrel{\text{def}}{=} (checkValid, r_5).PAProvider_3 \\
PAProvider_3 &\stackrel{\text{def}}{=} (stopSession, \top).PAProvider
\end{aligned}
$$

An alternative design is to have a stateless implementation, as below.

$$
\begin{aligned}
PAProvider \stackrel{\text{def}}{=}\ & (startSession, \top).PAProvider \\
& + (checkValid, r_5).PAProvider \\
& + (stopSession, \top).PAProvider
\end{aligned}
$$

We will contrast these two versions in our model analysis.

Model Component *WSComp*. The complete system is represented by some number of instances of the components interacting on their shared activities:

$$
\begin{aligned}
WSComp \stackrel{\text{def}}{=} \big(& (Customer[N_C] \underset{L_1}{\bowtie} WSProvider[N_{WSP}]) \\
& \underset{L_2}{\bowtie} WSConsumer[N_{WSC}]\big) \\
& \underset{L_3}{\bowtie} PAProvider[N_{PAP}]
\end{aligned}
$$

where the cooperation sets are

$$
\begin{aligned}
L_1 &= \{getSMS, getMap, get404\} \\
L_2 &= \{notify, locReq, locRes, locErr, sendMMS\} \\
L_3 &= \{startSession, checkValid, stopSession\}
\end{aligned}
$$

and N_C, N_{WSC}, N_{WSP} and N_{PAP} are the number of instances of *Customer*, *WSConsumer*, *WSProvider* and *PAProvider* respectively.

6.3 Performance Analysis of the Web Service Composition Case Study

In this section we carry out steady-state analysis on the Web Service Composition case study in order to tune the parameters of the system. To accomplish this task we use a modified version of the model in which the customer is explicitly modelled as a component of the system. The values for each rate are shown in Table 8.

Suppose that we want to design the system in such a way that it can handle 30 independent customers. The modeller may have constraints on some parameters such as the network delays because those are limited by the available technology. However, there are a number of degrees of freedom which let her vary, for example, the number of threads of control of the components of the system. The purpose is to deliver a satisfactory service in a cost-effective way. The simplest example of a cost function may be a linearly dependency on the number of copies of a component or the rate at which an activity is performed.

The graph in Fig. 12 shows the throughput of the *getMap* action as the number of customers varies between 1 and 30. Each line represents a given number of

Table 8. Parameters used in the performance analysis of the Web Service composition

parameter	*value*	*explanation*
r_1	0.0010	rate at which customers request maps
r_2	0.5	rate at which a session can be started
r_3	0.1	notification exchange between consumer and provider
r_4	0.1	rate at which requests for customer's location can be satisfied
r_5	0.05	rate at which the provider can check the validity of the incoming request
r_6	0.1	rate at which location information can be returned to the consumer
r_7	0.05	rate at which maps can be generated
r_8	0.02	rate at which MMS messages can be sent from provider to customer
r_9	$10.0 * r_8$	rate at which MMS messages can be sent via the Web Service

copies of the *WSProvider* component in the system. When the total number of customers is 30, two providers lead to a throughput which is twice as much as in the base system configuration with one provider only. However, as the number of provider increases the incremental benefit becomes less significant. In particular, the system with four copies is just 8.7% faster than the system with three. In the following we set to three the copies of *WSProvider*.

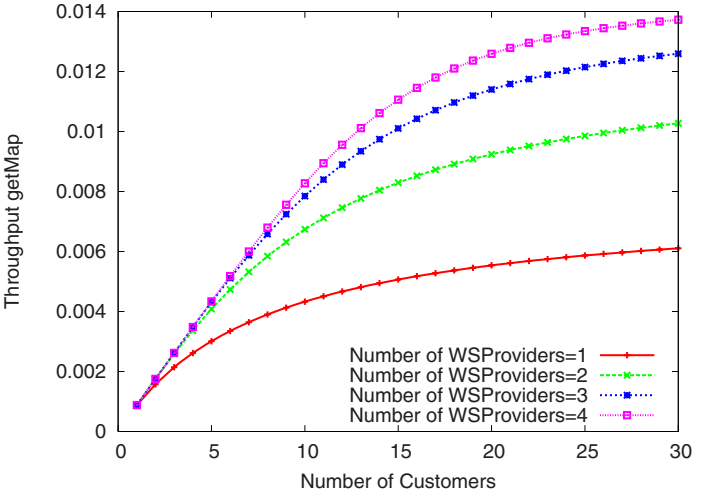


Fig. 12. Throughput of *getMap* for changes in the number of *WSProvider* and customers

In Fig. 13 is shown the effect that the rate at which the users initiate the request (r_1) has on the *getMap* throughput for different values of the copies of the *WSConsumer*. Every line starts to plateau at approximately $r_1 = 0.010$ following

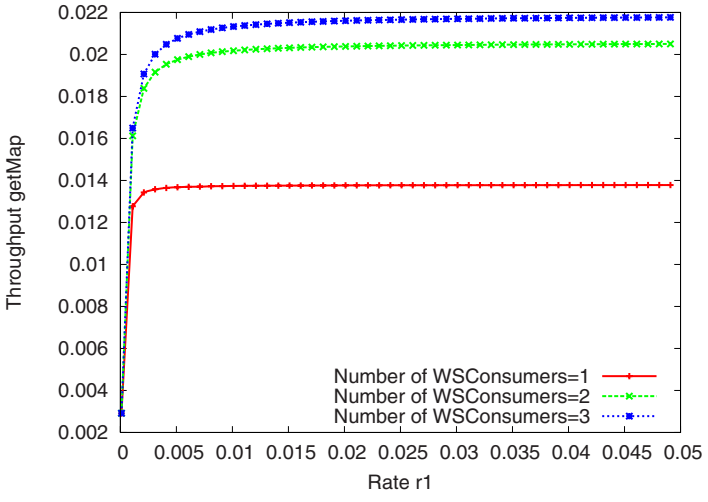


Fig. 13. Throughput of *getMap* for changes in the number of *WSConsumer* and r_1

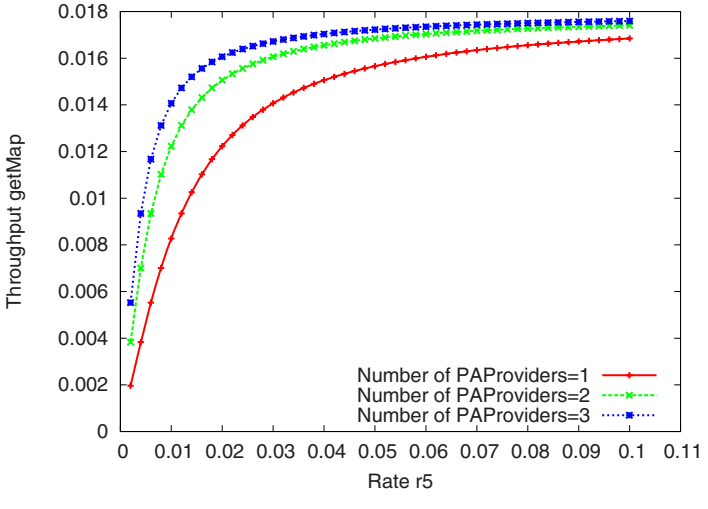


Fig. 14. Throughput of *getMap* for changes in the number of *PAProvider* and r_5

an initial sharp increase. This suggests that the system can guarantee satisfactory behaviour under the constraint that the users' request rate is below that threshold. In addition, the graph gives the modeller insights into the optimal number of operating threads of control of *WSConsumer*, which we believe is two as the additional third copy is not well matched by performance boost. Hence, in order to tune *PAProvider*—the remaining system component—we set *WSConsumer* to that value.

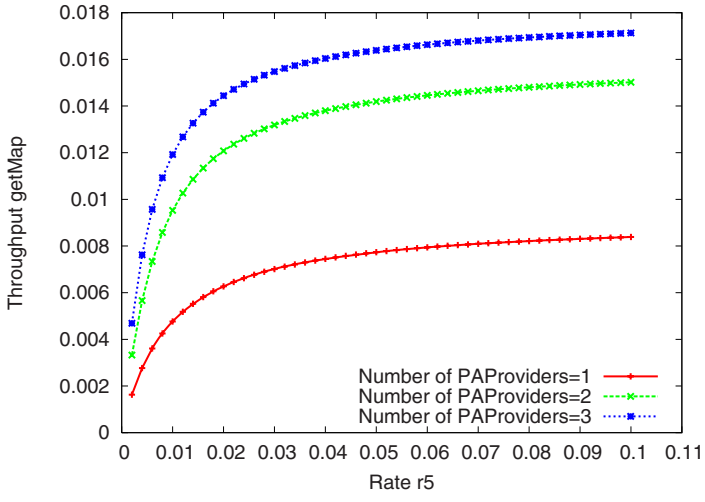


Fig. 15. Throughput of *getMap* for changes in the number of stateful *PAProvider* and r_5

The same approach can be applied to the optimisation of the number of copies of *PAProvider*. Here we are particularly interested in the overall impact of the rate at which the validity check is performed. Slower rates may mean more computationally expensive validation, whereas faster rates may involve less accuracy and lower security of the system. Such effects are measured in Fig. 14 where the *getMap* throughput is plotted against r_5 for different *PAProvider* pool sizes. A sharp increase followed by a constant levelling off suggests that optimal rate values lie on the left of the plateau, as faster rates do not improve the system considerably. As for the optimal number of copies of *PAProvider*, deploying two copies rather than one dramatically increases the quality of service of the overall system. With a similar approach as previously discussed, the modeller may want to consider the trade-off between the cost of adding a third copy and the throughput increase.

Evaluation of an alternative design of *PAProvider*. We conclude this section by showing how this model can be used to evaluate alternative designs of parts of the system. Here, we focus on *PAProvider* which has been originally modelled as a stateless component. Any of its services can be called at any point, the correctness of the system being guaranteed by implementation-specific constraints such as session identifiers being uniquely assigned to the clients and passed as parameters of the method calls.

Another design of a component which offers the same functionalities is that of a stateful provider. In PEPA such a service can be modelled as a sequential component with three local states (see above). This implementation has the consequence that there can never be at any point in time more than N_{WSP} *WSProvider* which have started a session with a *PAProvider*. This is because the provider has to release a previous session in order to start another one.

The graph in Fig. 15 measures the same metrics as in Fig. 14 when the stateful provider is employed. It shows that the incremental gain in adding more copies has become more noteworthy. However, the modeller may want to prefer the original version, as three copies of the stateful provider deliver about as much as the throughput of only one copy of the stateless implementation.

7 Advanced Topics

Like all state based modelling techniques, stochastic process algebra models are subject to the problem of state space explosion — the generated models may be intractable because of their size. A variety of techniques have been proposed for tackling this problem in the context of stochastic process algebra. Below we briefly discuss two of them:

- model reduction and model simplification via equivalence relations;
- fluid approximation of the state space.

7.1 Equivalence Relations and Model Manipulation

The state space explosion problem arises because although the compositionality of SPA can greatly aid model construction, in general the compositionality does not assist in the model solution and the resulting models may be too large to solve. This has led to research into how model simplification and aggregation techniques can be applied in the process algebra setting. Many such techniques are known in the context of Markov processes but are based on conditions phrased in terms of the process or its generator matrix. Moreover application of these techniques often relies on the expertise of the modeller. The challenge for SPA has been to define such model manipulation techniques in the context of the process algebra, in such a way that it can subsequently be applied automatically. Some significant results have been achieved in this area through the use of equivalence relations which provide the basis for comparing and manipulating models within a formal framework. Furthermore the compositionality of the process algebra allows these techniques to be applied to part of the model whilst maintaining the integrity of the model as a whole.

There have been two principal approaches to model manipulation in SPA:

model simplification: Here an equivalence relation is used in order to establish behavioural or observational equivalence *between models*. The aim is to replace one model by an equivalent one which is more desirable from a solution point of view. Once the desirable model has replaced the original, the underlying Markov process is generated as usual, associating one state with each node in the labelled transition system generated by the semantics. Equivalence relations which have been used in this way are *weak isomorphism* in PEPA [29,69], *Markovian bisimulation* and *weak bisimulation* in TIPP [70].

model aggregation: Here an equivalence relation is used in order to establish behavioural or observational equivalence *between states within a model*. The

aim is to use an alternative mapping from the labelled transition system, given by the semantics of the model, to the underlying Markov process. The equivalence relation is used to partition the nodes of the labelled transition system into equivalence classes. Then, instead of the usual one-to-one correspondence between nodes and states, one state in the underlying Markov process is associated with each equivalence class of nodes. The hope is that this will generate a Markov process with a smaller number of states. The equivalence relation which has been used in this way is variously called *strong equivalence* (PEPA) [29], *Markovian bisimulation* (TIPP) [71], and *extended Markovian bisimulation equivalence* (EMPA) [27].

Equivalence relations and model manipulations will be discussed in more detail in another chapter within this volume [72].

The basis of aggregation is the observation that it can be sufficient to consider the behaviour of one element within an equivalence class of elements who all behave in the same way. The simplest way in which such equivalence classes arise is if we have repeated instances of identical components within the model. For this case, for PEPA models we have developed an automatic method which generates the CTMC corresponding the equivalence classes, rather than the individual states, on-the-fly [47]. This relies on a *canonical representation* of states within the model which makes it clear syntactically when they are equivalent, while also keeping track of how many instances there are in each such equivalence class.

Since the static cooperation combinators remain unchanged in all states of a model, it is often convenient to represent the states in *vector form*. The state vector records one entry for each sequential component of the PEPA model. These components will be present in each derivative of the model, although they will change their local state or derivative. Thus the global state can be represented as a vector or sequence of local derivatives.

If a model contains equivalent components there may be multiple states within the model which exhibit the same behaviour and so we may aggregate the model. The derivation graph is then constructed in terms of equivalence classes of syntactic terms and this is used as the basis of the CTMC construction [47]. Canonicalisation involves reordering entries within the vector in a way that strong equivalence, the Markovian bisimulation of PEPA models, is respected, but which places elements within subvectors of equivalent components in lexicographical order. Further details can be found in [47].

7.2 Continuous State Space Approximation

Even with the use of aggregation some model still remain too large to be readily analysed using Markovian techniques. Recent work has considered a radically different approach to tackling the state space explosion problem when modelling with a process algebra such as PEPA [73]. The approach is based on two shifts from the usual perspective:

– Firstly, we do not aim to calculate the probability distribution over the entire state space of the model. We choose a more abstract state representation in

terms of state variables, quantifying the types of behaviour evident in the model.
– Secondly, we assume that these state variables are subject to *continuous* rather than *discrete* change.

Once these adjustments are made the system is amenable to efficient solution as a set of ordinary differential equations (ODEs), leading to the evaluation of transient, and in the limit, steady state measures.

State representation. As we have seen the usual state representation is in terms of the syntactic forms of the model expression, or when aggregation is applied, in terms of a canonical representation of an equivalence class of states.

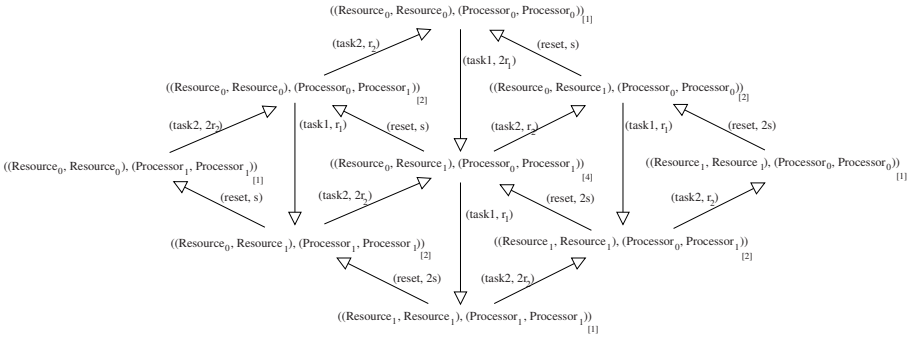
The work on continuous approximation proposes an alternative vector form for capturing the state information of models with repeated components. In the state vector form, even when the canonical representation is used there is one entry in the vector for each sequential component in the model. When the number of repeated components becomes large this can be prohibitively expensive in terms of storage. In the alternative vector form there is one entry for each local derivative of each type of component in the model. Two components have the same type if their derivation graphs are isomorphic. The entries in the vector are no longer syntactic terms representing the local derivative of the sequential component, but the *number* of components currently exhibiting this local derivative.

To clarify the distinction between the two vector forms consider the small example defined below, consisting of interacting processors and resources:

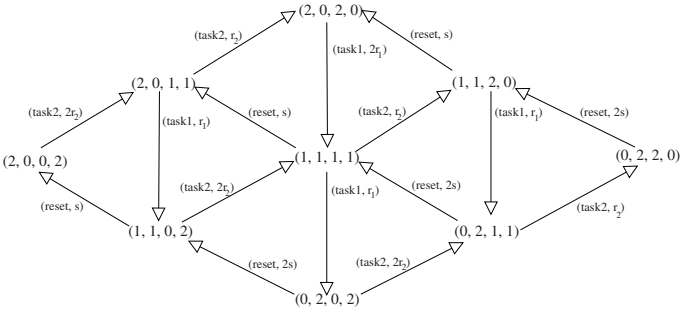
$$\begin{aligned} Processor_0 &\stackrel{\text{def}}{=} (task1, r_1).Processor_1 \\ Processor_1 &\stackrel{\text{def}}{=} (task2, r_2).Processor_0 \\ Resource_0 &\stackrel{\text{def}}{=} (task1, r_1).Resource_1 \\ Resource_1 &\stackrel{\text{def}}{=} (reset, s).Resource_0 \end{aligned}$$

$$(Resource_0 \parallel Resource_0) \underset{\{task1\}}{\bowtie} (Processor_0 \parallel Processor_0)$$

The canonical state vector form corresponding to this example with the given configuration is shown in Figure 16a). Here the initial state is represented explicitly as $((Resource_0, Resource_0), (Processor_0, Processor_0))_{\{task1\}}$. In contrast, in the numerical vector form, shown in Figure 16b), the initial state is $(2, 0, 2, 0)$ where the entries in the vector are counting the number of $Resource_0$, $Resource_1$, $Processor_0$, $Processor_1$ local derivatives respectively, exhibited in the current state. In the canonical state vector representation we record the number of elements in each equivalence class (shown in square brackets in Figure 16a). The total rate of the transitions between the canonical states is derived from this number of instances, the number of enabled activities and their relative probabilities. In the numerical state vector representation each vector is a single state and the rates of the transitions between states are derived directly from the vector and the activity rate, as explained below.



a) Aggregated state space in canonical form



b) Vector state space

Fig. 16. Illustrative example of contrasting state representations

In the current configuration of the model, with two instances of each component type, it is clear that the state vector form and the numerical vector form each have four elements, but if we consider a configuration with ten instances of each component type it becomes clear that the numerical form is much more compact.

The numerical vector form for an arbitrary PEPA model is defined as follows.

Definition 1 (Numerical Vector Form). *For an arbitrary PEPA model $\mathcal{M}$ with n component types $\mathcal{C}_i, i = 1, \ldots, n$, each with N_i distinct derivatives, the* numerical vector form *of $\mathcal{M}$, $\mathcal{V}(\mathcal{M})$, is a vector with $N = \sum_{i=1}^{n} N_i$ entries. The entry v_{i_j} records how many instances of the jth local derivative of component type $\mathcal{C}_i$ are exhibited in the current state.*

If there is a large number of instances of each component type the domain of values of each entry in $\mathcal{V}(\mathcal{M})$ is large. If K_i is the number of components of type $\mathcal{C}_i$ in the initial configuration of the model then each entry in the ith subvector will have domain $0, \ldots, K_i$.

The system is inherently discrete with the entries within the numerical vector form always being non-negative integers and always being incremented or decremented in steps of one. When the numbers of components are large these steps

are relatively small and we can approximate the behaviour by considering the movement between states to be continuous, rather than occurring in discontinuous jumps. In this case we can replace the discrete event system represented by the derivation graph of a PEPA process by a continuous model, represented by a set of coupled ordinary differential equations. The numerical vector form of state representation is an intermediate step to achieving that. Considering these states of the process and the activities which are enabled, and the states they lead to, we are able to construct an *activity matrix* which records the impact of each activity type on the number of each component type. From this the appropriate system of ODEs is derived (see [73] for details).

Small example revisited. Let us consider again the small example considered earlier, assuming now that there are large numbers of processors and resources:

$$Processor_0 \stackrel{\text{def}}{=} (task1, r_1).Processor_1$$
$$Processor_1 \stackrel{\text{def}}{=} (task2, r_2).Processor_0$$
$$Resource_0 \stackrel{\text{def}}{=} (task1, r_1).Resource_1$$
$$Resource_1 \stackrel{\text{def}}{=} (reset, s).Resource_0$$
$$(Resource_0 \parallel \cdots \parallel Resource_0) \underset{\{task1\}}{\bowtie} (Processor_0 \parallel \cdots \parallel Processor_0)$$

Let n_1 denote the number of $Processor_0$ entities, n_2 the number of $Processor_1$ entities, n_3 the number of Res_0 entities and n_4 the number of $Resource_1$ entities. The activity matrix corresponding the component definitions is shown in Fig. 17.

	$task_1$	$task_2$	$reset$	
$Processor_0$	-1	$+1$	0	n_1
$Processor_1$	$+1$	-1	0	n_2
$Resource_0$	-1	0	$+1$	n_3
$Resource_1$	$+1$	0	-1	n_4

Fig. 17. Activity matrix for the simple Processor-Resource model

From the matrix, we derive each differential equation in turn. For state variable n_i, consider row i. Each non-zero entry in the row will results in one term within the equation.

$$\frac{dn_1(t)}{dt} = -r_1 \min(n_1(t), n_3(t)) + r_2 n_2(t)$$
$$\frac{dn_2(t)}{dt} = r_1 \min(n_1(t), n_3(t)) - r_2 n_2(t)$$
$$\frac{dn_3(t)}{dt} = -r_1 \min(n_1(t), n_3(t)) + s n_4(t)$$
$$\frac{dn_4(t)}{dt} = r_1 \min(n_1(t), n_3(t)) - s n_4(t)$$

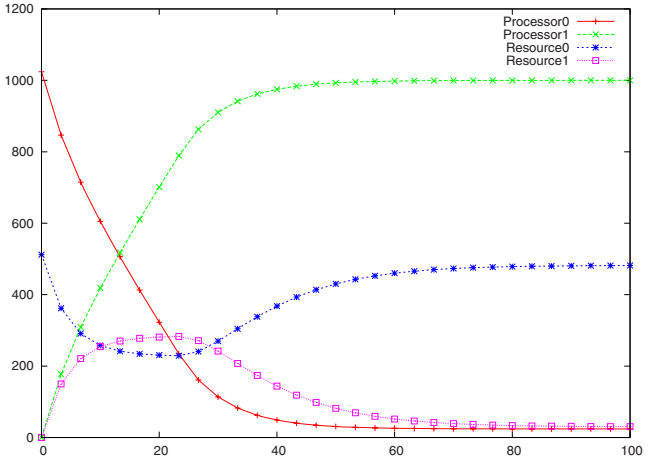


Fig. 18. Graph showing the changing numbers of copies of $Processor_0$, $Processor_1$, $Resource_0$ and $Resource_1$ as a function of time, obtained by numerically integrating the differential equations for this system. The values of the rates were $r_1 = 0.125$, $r_2 = 0.003$ and $s = 0.1$.

Note that the form of the system of equations is independent of the number of components included in the initial configuration of the model. The only impact of changing the number of instances of each component type is to alter the initial conditions. Thus, if there are initially 1024 processors, all starting in state $Processor_0$ and 512 resources, all of which start in state $Resource_0$, the initial conditions will be:

$$n_1(0) = 1024 \qquad n_2(0) = 0 \qquad n_3(0) = 512 \qquad n_4(0) = 0$$

Numerically integrating the differential equations for this system to generate a time series plot for the first 100 seconds of the system evolution starting from the above initial value problem produces the graph shown in Fig. 18.

8 Conclusions and Summary

In this tutorial we have described an algebraic description technique, based on a classical process algebra, and enhanced with timing information. This extension results in models which may be used to calculate performance measures as well as deduce functional properties of the system. Several interesting analysis techniques of SPA models including steady state, transient and response time analysis of the underlying CTMC have been discussed, together with an introduction to the tools which support these analysis techniques. We have demonstrated the approach on a number of small models as well as a more realistic example of a service-oriented architecture. Finally we outlined some more advanced topics related to SPA and highlighted some on-going work.

Acknowledgements

This work has been supported by the project EU FET-IST Global Computing 2 project SENSORIA ("Software Engineering for Service-Oriented Overlay Computers" (IST-3-016004-IP-09)). Jane Hillston is also supported by EPSRC Advanced Research Fellowship EP/c543696/01.

References

1. Herzog, U.: Formal description, time and performance analysis: A framework. Technical Report 15/90, IMMD VII, Friedrich-Alexander-Universität, Erlangen-Nürnberg, Germany (September 1990)
2. Holton, D.: A PEPA specification of an industrial production cell. 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) (December 1995) 542–551
3. Gilmore, S., Hillston, J., Holton, D., Rettelbach, M.: Specifications in Stochastic Process Algebra for a Robot Control Problem. International Journal of Production Research **34**(4) (1996) 1065–1080
4. Thomas, N., Hillston, J.: Using Markovian process algebra to specify interactions in queueing systems. Technical Report ECS-LFCS-97-373, Laboratory for Foundations of Computer Science, Department of Computer Science, The University of Edinburgh (1997)
5. Bowman, H., Bryans, J., Derrick, J.: Analysis of a multimedia stream using stochastic process algebra. In Priami, C., ed.: Sixth International Workshop on Process Algebras and Performance Modelling, Nice (September 1998) 51–69
6. Console, L., Picardi, C., Ribaudo, M.: Diagnosis and Diagnosability Analysis using PEPA. In: Proc. of 14^{th} European Conference on Artificial Intelligence, Berlin (August 2000) A longer version appeared in the Proc. of 11th Int. Workshop on Principles of Diagnosis (DX00), Morelia, Mexico, June 2000.
7. Hillston, J., Kloul, L.: Performance investigation of an on-line auction system. Concurrency and Computation: Practice and Experience **13** (2001) 23–41
8. Forneau, J., Kloul, L., Valois, F.: Performance modelling of hierarchical cellular networks using PEPA. Performance Evaluation **50**(2–3) (November 2002) 83–99
9. Brodo, L., Degano, P., Gilmore, S., Hillston, J., Priami, C.: Performance evaluation for global computation. In Priami, C., ed.: Global Computing: Programming environments, languages, security, and analysis of systems. Proceedings of the IST/FET International Workshop (GC 2003). Volume 2874 of LNCS., Rovereto, Italy, Springer-Verlag (February 2003) 229–253
10. Buchholtz, M., Gilmore, S., Hillston, J., Nielson, F.: Securing statically-verified communications protocols against timing attacks. Electr. Notes Theor. Comput. Sci. **128**(4) (2005) 123–143
11. Hillston, J., la Kloul, L., Mokhtari, A.: Towards a feasible active networking scenario. Telecommunication Systems **27**(2–4) (October 2004) 413–438
12. Fourneau, J.M., Kloul, L.: A precedence PEPA model for performance and reliability analysis. In Horváth, A., Telek, M., eds.: Formal Methods and Stochastic Models for Performance Evaluation: Third European Performance Engineering Workshop (EPEW 2006). Number 4054 in LNCS, Springer-Verlag (June 2006) 1–15

13. Duguid, A.: Coping with the parallelism of BitTorrent: Conversion of PEPA to ODEs in dealing with state space explosion. In Asarin, E., Bouyer, P., eds.: Formal Modeling and Analysis of Timed Systems, 4th International Conference, FORMATS 2006, Paris, France, September 25-27, 2006, Proceedings. Volume 4202 of Lecture Notes in Computer Science., Springer (2006) 156–170
14. Gilmore, S., Tribastone, M.: Evaluating the scalability of a web service-based distributed e-learning and course management system. In Mario Bravetti, M.T.N.n., Zavattaro, G., eds.: Third International Workshop on Web Services and Formal Methods (WS-FM'06). Volume 4184 of Lecture Notes in Computer Science., Vienna, Austria, Springer (2006) 156–170
15. Razafindralambo, T., Valois, F.: Performance evaluation of backoff algorithms in 802.11 ad-hoc networks. In: PE-WASUN '06: Proceedings of the 3rd ACM international workshop on Performance evaluation of wireless ad hoc, sensor and ubiquitous networks, New York, NY, USA, ACM Press (2006) 82–89
16. Razafindralambo, T., Valois, F.: Stochastic behavior study of backoff algorithms in case of hidden terminals. In: Proceedings of the 17th Annual IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC'06), IEEE Press (2006) 1–6
17. Milner, R.: Communication and Concurrency. Prentice-Hall (1989)
18. Hoare, C.: Communicating Sequential Processes. Prentice-Hall (1985)
19. Nicollin, X., Sifakis, J.: An Overview and Synthesis on Timed Process Algebras. In: Real-Time: Theory in Practice. Springer LNCS 600 (1991) 526–548
20. Moller, F., Tofts, C.: A Temporal Calculus for Communicating Systems. In Baeten, J., Klop, J., eds.: CONCUR'90. Volume 458 of LNCS., Springer-Verlag (August 1989) 401–415
21. Jou, C.C., Smolka, S.: Equivalences, Congruences and Complete Axiomatizations of Probabilistic Processes. In Baeten, J., Klop, J., eds.: CONCUR'90. Volume 458 of LNCS. Springer-Verlag (August 1990) 367–383
22. Edinburgh Concurrency Workbench. http://homepages.inf.ed.ac.uk/perdita/cwb/
23. Hennessy, M., Milner, R.: On observing nondeterminism and concurrency. In de Bakker, J.W., van Leeuwen, J., eds.: Proceedings 7^{th} ICALP, *Noordwijkerhout*. Volume 85 of Lecture Notes in Computer Science., Springer-Verlag (July 1980) 299–309
24. Kozen, D.: Results on the propositional μ-calculus. Theoretical Computer Science **27**(3) (1983) 333–354
25. Götz, N., Herzog, U., Rettelbach, M.: TIPP—a language for timed processes and performance evaluation. Technical Report 4/92, IMMD7, University of Erlangen-Nürnberg, Germany (November 1992)
26. Bernardo, M., Gorrieri, R., Donatiello, L.: MPA: A Stochastic Process Algebra. Technical Report UBLCS-94-10, Laboratory of Computer Science, University of Bologna (May 1994)
27. Bernardo, M., Gorrieri, R.: A Tutorial on EMPA: A Theory of Concurrent Processes with Nondeterminism, Priorities, Probabilities and Time. Theoretical Computer Science **to appear** (1998)
28. Hillston, J.: PEPA - Performance Enhanced Process Algebra. Technical report, Dept. of Computer Science, University of Edinburgh (March 1993)
29. Hillston, J.: A Compositional Approach to Performance Modelling. PhD thesis, Department of Computer Science, University of Edinburgh (April 1994) CST-107-94.

30. Strulo, B.: Process Algebra for Discrete Event Simulation. PhD thesis, Imperial College (1993)
31. Priami, C.: Stochastic π-Calculus. The Computer Journal **38**(6) (1995)
32. Hermanns, H.: Interactive Markov Chains: The Quest for Quantified Quality. Volume 2428 of LNCS. Springer (2002)
33. D'Argenio, P., Hermanns, H., Katoen, J.P., Klaren, R.: MoDeST: A modelling language for stochastic timed systems. In: Process Algebra and Probabilistic Methods, Springer-Verlang LNCS 2165 (2001) 87–104
34. Bravetti, M., Gorrieri, R.: The theory of Interactive Generalized Semi-Markov Processes. Theoretical Computer Science **282**(1) (June 2002) 5–32
35. Bravetti, M., Bernardo, M., Gorrieri, R.: From EMPA to GSMPA: Allowing for general distributions. In Brinksma, E., Nymeyer, A., eds.: Proc. of the 5th Int. Workshop on Process Algebras and Performance Modeling (PAPM '97). (1997) 17–33
36. Rettelbach, M.: Probabilistic Branching in Markovian Process Algebras. The Computer Journal **38**(6) (1995) Special Issue: Proc. of 3rd Process Algebra and Performance Modelling Workshop.
37. Hermanns, H., Rettelbach, M., Weiß, T.: Formal Characterisation of Immediate Actions in SPA with Nondeterministic Branching. The Computer Journal **38**(6) (1995) Special Issue: Proc. of 3rd Workshop on Process Algebras and Performance Modelling.
38. Hillston, J.: The nature of synchronisation. In Herzog, U., Rettelbach, M., eds.: Proceedings of the Second International Workshop on Process Algebras and Performance Modelling, Erlangen (November 1994) 51–70
39. Ribaudo, M.: Understanding Stochastic Process Algebras via their Stochastic Petri Net Semantics. In Herzog, U., Rettelbach, M., eds.: Proc. of 2nd Process Algebra and Performance Modelling Workshop. (1994)
40. Bradley, J.: Towards Reliable Modelling with Stochastic Process Algebras. PhD thesis, Department of Computer Science, University of Bristol (1999)
41. Hillston, J.: A Compositional Approach to Performance Modelling. Cambridge University Press (1996)
42. Clark, G., Gilmore, S., Hillston, J., Ribaudo, M.: Exploiting modal logic to express performance measures. In Haverkort, B., Bohnenkamp, H., Smith, C., eds.: Computer Performance Evaluation: Modelling Techniques and Tools, Proceedings of the 11th International Conference. Number 1786 in LNCS, Schaumburg, Illinois, USA, Springer-Verlag (March 2000) 211–227
43. Dempster, E.W., Tomov, N.T., Lü, J., Pua, C.S., Williams, M.H., Burger, A., Taylor, H., Broughton, P.: Verifying a performance estimator for parallel DBMSs. In: Proceedings of EuroPar (EuroPar'98). (September 1998)
44. Bouzeghoub, M., Kloul, L., Mokhtari, A.: A new active network framework based on active rules. Technical Report 2002/21, PRiSM, Université de Versailles (2002)
45. Hillston, J., Kloul, L., Mokhtari, A.: Active nodes performance analysis using PEPA. [74] 244–256
46. Hillston, J., Kloul, L., Mokhtari, A.: Towards a feasible active networking scenario. Telecommunication Systems **27**(2–4) (2004) 413–438
47. Gilmore, S., Hillston, J., Ribaudo, M.: An efficient algorithm for aggregating PEPA models. IEEE Transactions on Software Engineering **27**(5) (May 2001) 449–464
48. Console, L., Picardi, C., Ribaudo, M.: Diagnosis and Diagnosability Analysis using Process Algebras. In: Proc. of 11th Int. Workshop on Principles of Diagnosis (DX00), Morelia, Mexico (June 2000)

49. Eclipse.org home. http://www.eclipse.org
50. Eclipse Modeling Framework. http://www.eclipse.org/home
51. Matrix Toolkit for Java. http://rs.cipr.uib.no/mtj/
52. Bradley, J., Dingle, N., Gilmore, S., Knottenbelt, W.: Derivation of passage-time densities in PEPA models using IPC: The Imperial PEPA Compiler. In Kotsis, G., ed.: Proceedings of the 11th IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunications Systems, University of Central Florida, IEEE Computer Society Press (October 2003) 344–351
53. Knottenbelt, W.: Generalised Markovian analysis of timed transition systems. Master's thesis, University of Cape Town (1996)
54. Bradley, J., Dingle, N., Gilmore, S., Knottenbelt, W.: Extracting passage times from PEPA models with the HYDRA tool: A case study. [74] 79–90
55. Argent-Katwala, A., Bradley, J., Dingle, N.: Expressing performance requirements using regular expressions to specify stochastic probes over process algebra models. In: Proceedings of the Fourth International Workshop on Software and Performance, Redwood Shores, California, USA, ACM Press (January 2004) 49–58
56. PRISM. http://www.cs.bham.ac.uk/~dxp/prism/index.php
57. Hermanns, H., Meyer-Kayser, J., Siegle, M.: Multi-terminal binary decision diagrams to represent and analys e continuous time markov chains. In: Proc. of 3rd Intl. Workshop on the Numerical Solution of Mark ov Chains. (1999) 188–207
58. Gilmore, S., Kloul, L.: A unified tool for performance modelling and prediction. In S. Anderson, B.L., Felici, M., eds.: Proceedings of the 22nd International Conference on Computer Safety, Reliability and Security (SAFECOMP 2003). Volume 2788 of LNCS., Springer-Verlag (2003) 179–192
59. Gilmore, S., Hillston, J., Kloul, L., Ribaudo, M.: Software performance modelling using PEPA nets. In: Proceedings of the Fourth International Workshop on Software and Performance, Redwood Shores, California, USA, ACM Press (January 2004) 13–24
60. Clark, G., Courtney, T., Daly, D., Deavours, D., Derisavi, S., Doyle, J.M., Sanders, W.H., Webster, P.: The Möbius modeling tool. In: Proc. of 9th Int. Workshop on Petri Nets and Performance Models, Aachen, Germany (September 2001) 241–250
61. Clark, G., Sanders, W.: Implementing a stochastic process algebra within the Möbius modeling framework. In de Alfaro, L., Gilmore, S., eds.: Proceedings of the first joint PAPM-PROBMIV Workshop. Volume 2165 of Lecture Notes in Computer Science., Aachen, Germany, Springer-Verlag (September 2001) 200–215
62. : TwoTowers 5.1. http://www.sti.uniurb.it/bernardo/twotowers/
63. Hermanns, H., Mertsiotakis, V.: A Stochastic Process Algebra Based Modelling Tool. In: Proc. of the 11th UK Performance Engineering Workshop for Computer and Telecommunication Systems, Springer (1995)
64. Hermanns, H., Herzog, U., Klehmet, U., Mertsiotakis, V., Siegle, M.: Compositional performance modelling with the TIPPtool. In: Proc. of 10th International Conference on Modelling Techniques and Tools for Computer Performance Evaluation. Volume 1469 of LNCS., Palma de Mallorca, Springer-Verlag (1998)
65. Cleaveland, W., Sims, S.: The NCSU Concurrency Workbench. In: Proc. of Int. Conf. on Computer Aided Verification (CAV'96). Volume 1102 of LNCS., Springer-Verlag (1996) 394–397
66. Stewart, W.: Introduction to the Numerical Solution of Markov Chains. Princeton University Press (1994)

67. Bohnenkamp, H., Courtney, T., Daly, D., Derisavi, S., Hermanns, H., Katoen, J.P., Klaren, R., Lamb, V., Sanders, W.: On integrating the MÖBIUS and MODEST modeling tools. In: Proceedings of the 2003 International Conference on Dependable Systems and Networks (DSN'03), IEEE Computer Society Press (2003)
68. Garavel, H., Lang, F., Mateescu, R.: An overview of CADP 2001. Technical Report RT-254, INRIA (2001)
69. Clark, G.: An Extended Weak Isomorphism for Model Simplification. In Brinksma, E., Nymeyer, A., eds.: Proc. of 5th Process Algebra and Performance Modelling Workshop. (1997)
70. Mertsiotakis, V.: Approximate Analysis Methods for Stochastic Process Algebras. PhD thesis, Universität Erlangen–Nürnberg, Martensstraße 3, 91058 Erlangen (September 1998)
71. Hermanns, H., Rettelbach, M.: Syntax, Semantics, Equivalences and Axioms for MTIPP. In Herzog, U., Rettelbach, M., eds.: Proc. of 2nd Process Algebra and Performance Modelling Workshop. (1994)
72. Bernardo, M.: Behavioural equivalences and model manipulations. In Bernardo, M., Hillston, J., eds.: Formal Methods for Performance Evaluation. LNCS. Springer (2007)
73. 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 (September 2005) 33–43
74. Jarvis, S., ed.: Proceedings of the Ninteenth UK Performance Engineering Workshop, University of Warwick (July 2003)

A Survey of Markovian Behavioral Equivalences

Marco Bernardo

Università di Urbino "Carlo Bo" – Italy
Istituto di Scienze e Tecnologie dell'Informazione

Abstract. Markovian behavioral equivalences are a means to relate and manipulate the formal descriptions of systems with an underlying CTMC semantics. There are three fundamental approaches to their definition: bisimilarity, testing, and trace. In this paper we survey the major results appeared in the literature about Markovian bisimilarity, Markovian testing equivalence, and Markovian trace equivalence. The objective is to compare these equivalences with respect to a number of criteria such as their discriminating power, the exactness of the CTMC-level aggregations they induce, the achievement of the congruence property, the existence of sound and complete axiomatizations, the existence of logical characterizations, and the existence of efficient verification algorithms.

1 Introduction

Performance-oriented notations provide the designer with the capability of building performance-aware system models, which can be used in the early development stages to predict the satisfiability of certain performance requirements as well as to choose among alternative designs on the basis of their expected QoS guarantees. These notations range from more theoretical ones – like queueing networks [38], stochastic Petri nets [1], and stochastic process algebras [32] – to more applicative ones – like formal modeling languages (MODEST [12]), architectural description languages (ÆMILIA [5]), coordination languages (STOCKLAIM [22]), and object-oriented modeling languages (UML SPT/MARTE [48]).

An important feature shared by most of the performance-oriented notations mentioned above is that of providing behavioral models of the systems under construction. Given two such models, establishing whether they are equivalent amounts to establishing whether the systems they represent behave the same. What is needed is thus a notion of behavioral equivalence. This would be useful not only to relate models that are syntactically different, but also to manipulate models in a way that preserves their functional and performance properties.

Among the various proposals appeared in the literature [25], there are three fundamental approaches to the definition of behavioral equivalences: bisimilarity [43,41], testing [21], and trace [33]. In the first approach, two models are considered to be equivalent if they are able to mimic each other's behavior stepwise. In the second approach, two models are considered to be equivalent if an external observer cannot distinguish between them, with the only way for the observer to compare their behaviors being to interact with them by means of

M. Bernardo and J. Hillston (Eds.): SFM 2007, LNCS 4486, pp. 180–219, 2007.

tests and look at their reactions. In the third approach, similarly to traditional automata theory, two models are considered to be equivalent if they are able to perform the same sequences of activities.

These three approaches, originally conceived for reasoning about functional aspects, have been subsequently extended to deal with non-functional aspects. As far as performance aspects are concerned, research has mainly concentrated on models of systems with an underlying continuous-time Markov chain (CTMC) semantics. The reason is that, due to their memoryless property, exponential distributions result in a simpler mathematical treatment without sacrificing expressiveness. In fact, besides being adequate for many real-life phenomena (like arrival processes and failure events), exponential distributions provide the most appropriate stochastic approximation if only the average duration of an activity is known, and proper combinations of them (called phase-type distributions) can approximate most of general distributions arbitrarily closely.

This has resulted in the development of the Markovian versions of bisimilarity, testing equivalence, and trace equivalence, which will be surveyed in this paper by recalling from [32,15,31,14,18,4,23,10,30,8,7,9,49] their properties.

Although behavioral equivalences abstract from specific kinds of models, most of their properties can be better investigated and understood in a process algebraic framework [41,33,2,6]. For this reason, the three Markovian behavioral equivalences mentioned above will be defined in this paper over Markovian process calculi. Many such calculi have been proposed in the literature, like TIPP [27], PEPA [32], MPA [16], EMPA$_{\rm gr}$ [10], Sπ [44], IMC [30], and PIOA [46]. They differ for the action representation – durational actions (TIPP, PEPA, MPA, EMPA$_{\rm gr}$, Sπ, PIOA) vs. instantaneous actions separated from time passing (IMC) – as well as for the action synchronization discipline – symmetric (TIPP, MPA, Sπ, IMC), asymmetric (EMPA$_{\rm gr}$, PIOA), or both (PEPA).

In this paper we shall start with a sequential Markovian process calculus (SMPC) with durational actions, which generates all the finite CTMCs with as few operators as possible: the null term, the action prefix operator, the alternative composition operator, and recursion. Then we shall add a parallel composition operator governed by an asymmetric action synchronization discipline, thus resulting in a concurrent Markovian process calculus (CMPC). We shall also address some syntax variations – like the inclusion of rewards, nondeterminism, and prioritized/weighted immediate actions – in order to present some useful variants of the considered equivalences.

Markovian bisimilarity, Markovian testing equivalence, and Markovian trace equivalence will be compared with respect to the following criteria:

1. *Discriminating power.* The three Markovian behavioral equivalences, together with some variants of Markovian trace equivalence, will be ordered according to a finer-than/coarser-than relation, thus providing information about the linear-time/branching-time spectrum in the Markovian case. As we shall see, similarly to what happens in the probabilistic case [36,35], the Markovian spectrum is more condensed than the nondeterministic one [25].

2. *Exactness.* Each of the three Markovian behavioral equivalences induces an aggregation at the CTMC level. In general, a CTMC aggregation is exact whenever the transient/stationary probability of being in a macrostate of an aggregated CTMC is the sum of the transient/stationary probabilities of being in one of the constituent microstates of the original CTMC. This guarantees the preservation of the performance characteristics when going from the original CTMC to the aggregated one. As we shall see, all the three Markovian behavioral equivalences induce exact aggregations. In other words, the three approaches – bisimilarity, testing, trace – to the definition of behavioral equivalences are not only intuitively appropriate from the functional viewpoint, but also meaningful for performance evaluation purposes.
3. *Congruence.* A behavioral equivalence that is a congruence with respect to the typical process algebraic operators is particularly helpful in practice, as it supports compositional reasoning. This enables the compositional reduction of the model state space. As we shall see, Markovian bisimilarity and Markovian testing equivalence are congruences, whereas Markovian trace equivalence – unlike the nondeterministic case but similarly to the probabilistic one [36] – is not a congruence with respect to parallel composition.
4. *Axiomatization.* The axiomatization of a behavioral equivalence elucidates the fundamental equational laws on which the equivalence relies. This equational characterization is thus useful to understand what models can be related by the equivalence. Whenever it is sound and complete, the axiomatization gives rise to the specific rules of a deduction system – including reflexivity, symmetry, transitivity, and substitutivity – that can be exploited as a rewriting system to syntactically manipulate the models in a way that is consistent with the equivalence.
5. *Logical characterization.* The modal/temporal logic characterization of a behavioral equivalence shows what behavioral properties are preserved by the equivalence. This can be exploited to provide diagnostic information that explains why two models are not equivalent. As we shall see, Markovian bisimilarity preserves branching-time properties, while Markovian trace equivalence preserves linear-time properties.
6. *Verification complexity.* In order to be applicable in practice, a behavioral equivalence must be equipped with an efficient verification algorithm. As we shall see, not only Markovian bisimilarity but also Markovian testing and trace equivalences – unlike the nondeterministic case [37] but similarly to the probabilistic one [35] – are all decidable in polynomial time.

This paper is organized as follows. In Sect. 2 we introduce the syntax and the semantics for SMPC and CMPC. In Sect. 3 we study Markovian bisimilarity. In Sect. 4 we address some of its variants that include rewards, nondeterminism, and prioritized/weighted immediate actions. In Sect. 5 we present Markovian testing equivalence. In Sect. 6 we illustrate Markovian trace equivalence together with some other trace-based Markovian behavioral equivalences. Finally, in Sect. 7 we summarize the comparison of the Markovian behavioral equivalences based on the criteria explained above and we discuss some open problems.

2 Basic Markovian Process Calculi

In this section we introduce two basic Markovian process calculi with durational actions. The first one is a sequential Markovian process calculus (SMPC) that generates all the finite CTMCs with as few operators as possible: the null term, the action prefix operator, the alternative composition operator, and recursion. The second one is a concurrent Markovian process calculus (CMPC) as it additionally includes a parallel composition operator governed by an asymmetric action synchronization discipline. Then we introduce some notation concerned with the exit rates of the process terms and the attributes associated with their computations.

2.1 Syntax and Semantics for SMPC

In SMPC every action is durational, hence it is represented as a pair $<a, \lambda>$, where $a \in \mathit{Name}$ is the name of the action while $\lambda \in \mathbb{R}_{>0}$ is the rate of the exponential distribution quantifying the duration of the action. The average duration of an exponentially timed action is equal to the inverse of its rate.

Whenever several exponentially timed actions are enabled, the race policy is adopted, hence the fastest action is the one that is executed. As a consequence of this generative [26] selection mechanism, the execution probability of any enabled exponentially timed action is proportional to its rate and the average sojourn time associated with a process term is exponentially distributed with rate given by the sum of the rates of the actions enabled by the term.

We denote by $\mathit{Act}_{\mathrm{S}} = \mathit{Name} \times \mathbb{R}_{>0}$ the set of the actions of SMPC. Unlike standard process theory, where a distinguished symbol τ is used as the name of the invisible action, here we assume that all the actions are visible.

Definition 1. *The set $\mathcal{L}_{\mathrm{S}}$ of the process terms of SMPC is generated by the following syntax:*

$$\begin{array}{l} P ::= \underline{0} \\ \quad \mid\ <a, \lambda>.P \\ \quad \mid\ P + P \\ \quad \mid\ X \\ \quad \mid\ \mathit{rec}\, X : P \end{array}$$

where X is a process variable. We denote by $\mathcal{P}_{\mathrm{S}}$ the set of the closed and guarded process terms of SMPC. ■

The semantics for SMPC is given by a state-transition model that can be defined in the usual operational style. However, unlike nondeterministic process calculi, idempotency no longer holds. In fact, a term like $<a, \lambda>.P + <a, \lambda>.P$ is not the same as $<a, \lambda>.P$, as the average sojourn time associated with the latter, i.e. $1/\lambda$, is twice the average sojourn time associated with the former, i.e. $1/(\lambda + \lambda)$. To keep the two terms distinct at the semantic level, it is necessary to take into account the multiplicity of each transition, intended as the number of different proofs for the derivation of the transition.

Therefore, the behavior of each process term $P \in \mathcal{P}_{\mathrm{S}}$ is given by a labeled multitransition system $[\![P]\!]$, whose states correspond to process terms and whose transitions – each of which has a multiplicity – are labeled with actions. From such a labeled multitransition system the CTMC underlying the process term can easily be retrieved by (i) discarding the action names from the transition labels and (ii) collapsing all the transitions between any two states into a single transition whose rate is the sum of the rates of the original transitions.

We now provide the semantic rules for the various operators of SMPC:

- Null term: $\underline{0}$ cannot execute any action, hence the corresponding labeled multitransition system is just a state with no transitions.
- Exponentially timed action prefix: $<a, \lambda>.P$ can execute an action of name a and rate λ and then behaves as P:

$$\boxed{<a, \lambda>.P \stackrel{a,\lambda}{\longrightarrow} P}$$

- Alternative composition: $P_1 + P_2$ behaves as either P_1 or P_2 depending on whether P_1 or P_2 executes an action first:

$$\boxed{\frac{P_1 \stackrel{a,\lambda}{\longrightarrow} P'}{P_1 + P_2 \stackrel{a,\lambda}{\longrightarrow} P'} \qquad \frac{P_2 \stackrel{a,\lambda}{\longrightarrow} P'}{P_1 + P_2 \stackrel{a,\lambda}{\longrightarrow} P'}}$$

- Recursion: $rec\, X : P$ behaves as P after replacing every occurrence of X with $rec\, X : P$:

$$\boxed{\frac{P\{(rec\, X : P)/X\} \stackrel{a,\lambda}{\longrightarrow} P'}{rec\, X : P \stackrel{a,\lambda}{\longrightarrow} P'}}$$

2.2 Syntax and Semantics for CMPC

CMPC extends SMPC with a parallel composition operator governed by an asymmetric action synchronization discipline, which is enforced on an explicit set of action names and makes use of passive actions. Multiway synchronizations are allowed provided that they involve at most one exponentially timed action, with all the other actions being passive.

A passive action is of the form $<a, *_w>$, where $w \in \mathbb{R}_{>0}$ is called weight and is used to quantify choices among passive actions with the same name. Every passive action has a duration that will become specified upon synchronization with an exponentially timed action having the same name.

Whenever several passive actions are enabled, the reactive [26] preselection policy is adopted. This means that, within every set of enabled passive actions with the same name, each such action is given an execution probability proportional to its weight. The choice between two enabled passive actions having different names is instead nondeterministic.

We denote by $Act_{\mathrm{C}} = Name \times Rate$ the set of the actions of CMPC, where $Rate = \mathbb{R}_{>0} \cup \{*_w \mid w \in \mathbb{R}_{>0}\}$ is the set of the action rates (ranged over by $\tilde{\lambda}$). As for SMPC, we assume that all the actions are visible.

Definition 2. *The set $\mathcal{L}_{\rm C}$ of the process terms of CMPC is generated by the following syntax:*

$$\begin{array}{l} P ::= \underline{0} \\ \quad\mid\ <a,\lambda>.P \\ \quad\mid\ <a,*_w>.P \\ \quad\mid\ P + P \\ \quad\mid\ P \parallel_S P \\ \quad\mid\ X \\ \quad\mid\ rec\ X : P \end{array}$$

where $S \subseteq Name$ and X is a process variable. We denote by $\mathcal{P}_{\rm C}$ the set of the closed and guarded process terms of CMPC. ■

Due to the memoryless property of exponential distributions and the fact that the probability that two concurrent exponentially timed actions terminate simultaneously is zero, the semantics for the parallel composition operator can be defined in the usual interleaving style like in the nondeterministic case. In fact, term $<a,\lambda>.\underline{0} \parallel_\emptyset <b,\mu>.\underline{0}$ behaves exactly like term $<a,\lambda>.<b,\mu>.\underline{0} + <b,\mu>.<a,\lambda>.\underline{0}$ as the execution of an exponentially timed action can be thought of as being started in the last state in which the action is enabled.

We now provide the semantic rules for the additional operators of CMPC:

– Passive action prefix: $<a,*_w>.P$ can execute a passive action of name a and weight w and then behaves as P:

$$<a,*_w>.P \stackrel{a,*_w}{\longrightarrow} P$$

– Parallel composition: $P_1 \parallel_S P_2$ behaves as P_1 in parallel with P_2 as long as actions are executed whose names do not belong to S:

$$\frac{P_1 \stackrel{a,\tilde{\lambda}}{\longrightarrow} P_1' \quad a \notin S}{P_1 \parallel_S P_2 \stackrel{a,\tilde{\lambda}}{\longrightarrow} P_1' \parallel_S P_2} \qquad \frac{P_2 \stackrel{a,\tilde{\lambda}}{\longrightarrow} P_2' \quad a \notin S}{P_1 \parallel_S P_2 \stackrel{a,\tilde{\lambda}}{\longrightarrow} P_1 \parallel_S P_2'}$$

Generative-reactive synchronizations are forced between any exponentially timed action executed by one term and any passive action executed by the other term that have the same name belonging to S:

$$\frac{P_1 \stackrel{a,\lambda}{\longrightarrow} P_1' \quad P_2 \stackrel{a,*_w}{\longrightarrow} P_2' \quad a \in S}{P_1 \parallel_S P_2 \stackrel{a,\lambda \cdot \frac{w}{weight(P_2,a)}}{\longrightarrow} P_1' \parallel_S P_2'}$$

$$\frac{P_1 \stackrel{a,*_w}{\longrightarrow} P_1' \quad P_2 \stackrel{a,\lambda}{\longrightarrow} P_2' \quad a \in S}{P_1 \parallel_S P_2 \stackrel{a,\lambda \cdot \frac{w}{weight(P_1,a)}}{\longrightarrow} P_1' \parallel_S P_2'}$$

while reactive-reactive synchronizations are forced between any two passive actions executed by the two terms that have the same name belonging to S:

$$\frac{P_1 \xrightarrow{a,*_{w_1}} P_1' \qquad P_2 \xrightarrow{a,*_{w_2}} P_2' \qquad a \in S}{P_1 \parallel_S P_2 \xrightarrow{a,* \frac{w_1}{weight(P_1,a)} \cdot \frac{w_2}{weight(P_2,a)} \cdot (weight(P_1,a)+weight(P_2,a))} P_1' \parallel_S P_2'}$$

where the weight of a process term P with respect to the passive actions of name a that P enables is defined as follows:

$$weight(P,a) = \sum\{\!|\, w \in \mathbb{R}_{>0} \mid \exists P' \in \mathcal{P}_{\rm C}.\, P \xrightarrow{a,*_w} P' \,|\!\}$$

We point out that the CTMC underlying a process term in $\mathcal{P}_{\rm C}$ can be retrieved only if its labeled multitransition system has no passive transitions. In this case we say that the process term is performance closed. We denote by $\mathcal{P}_{\rm C,pc}$ the set of the performance closed process terms of $\mathcal{P}_{\rm C}$. Note that $\mathcal{P}_{\rm S,pc} = \mathcal{P}_{\rm S}$.

2.3 Exit Rates and Computations of Process Terms

The Markovian behavioral equivalences that we shall define over SMPC and CMPC are based on concepts like the exit rates of the process terms and the traces, the probabilities, and the durations of their computations. Since these concepts will be used several times in the paper, we collect in this section the related notation.

The exit rate of a process term is the rate at which it is possible to leave the term. We distinguish among the rate at which the process term can execute actions of a given name that lead to a given set of terms, the total rate at which the process term can execute actions of a given name, and the total exit rate of the process term. The latter is the sum of the rates of all the actions that the process term can execute, and coincides with the reciprocal of the average sojourn time in the CTMC-level state corresponding to the process term whenever the process term is performance closed.

Since there are two kinds of actions – exponentially timed and passive – we consider a two-level definition of each variant of exit rate, where level 0 corresponds to exponentially timed actions and level -1 corresponds to passive actions.

Definition 3. *Let $P \in \mathcal{P}_{\rm C}$, $a \in Name$, $l \in \{0,-1\}$, and $C \subseteq \mathcal{P}_{\rm C}$. The exit rate of P when executing actions of name a and level l that lead to C is defined through the following non-negative real function:*

$$rate(P,a,l,C) = \begin{cases} \sum\{\!|\, \lambda \in \mathbb{R}_{>0} \mid \exists P' \in C.\, P \xrightarrow{a,\lambda} P' \,|\!\} & \text{if } l = 0 \\ \sum\{\!|\, w \in \mathbb{R}_{>0} \mid \exists P' \in C.\, P \xrightarrow{a,*_w} P' \,|\!\} & \text{if } l = -1 \end{cases}$$

where each summation is taken to be zero whenever its multiset is empty. ■

Definition 4. *Let $P \in \mathcal{P}_{\rm C}$ and $l \in \{0, -1\}$. The total exit rate of P at level l is defined through the following non-negative real function:*

$$\boxed{rate_{\rm t}(P, l) = \sum_{a \in Name} rate(P, a, l, \mathcal{P}_{\rm C})}$$

where $rate(P, a, l, \mathcal{P}_{\rm C})$ is the total exit rate of P with respect to a at level l. ■

A computation of a process term is a sequence of transitions that can be executed starting from the state corresponding to the term. The length of a computation is given by the number of transitions occurring in it. We say that two computations are independent of each other if it is not the case that one of them is a proper prefix of the other one. In the following, we denote by $\mathcal{C}_{\rm f}(P)$ and $\mathcal{I}_{\rm f}(P)$ the multisets of the finite-length computations and independent computations of $P \in \mathcal{P}_{\rm C}$. Below we inductively define the trace, the execution probability, the average duration, and the duration distribution of an element of $\mathcal{C}_{\rm f}(P)$, using symbol "$\circ$" to denote the sequence concatenation operator.

Definition 5. *Let $P \in \mathcal{P}_{\rm C}$ and $c \in \mathcal{C}_{\rm f}(P)$. The trace associated with the execution of c is the sequence of the action names labeling the transitions of c, which is defined by induction on the length of c through the following $Name^*$-valued function:*

$$\boxed{trace(c) = \begin{cases} \varepsilon & \text{if } length(c) = 0 \\ a \circ trace(c') & \text{if } c \equiv P \xrightarrow{a,\tilde{\lambda}} c' \end{cases}}$$

where ε is the empty trace. ■

Definition 6. *Let $P \in \mathcal{P}_{\rm C,pc}$ and $c \in \mathcal{C}_{\rm f}(P)$. The probability of executing c is the product of the execution probabilities of the transitions of c, which is defined by induction on the length of c through the following $\mathbb{R}_{]0,1]}$-valued function:*

$$\boxed{prob(c) = \begin{cases} 1 & \text{if } length(c) = 0 \\ \frac{\lambda}{rate_{\rm t}(P,0)} \cdot prob(c') & \text{if } c \equiv P \xrightarrow{a,\lambda} c' \end{cases}}$$

We also define the probability of executing a computation of C as:

$$\boxed{prob(C) = \sum_{c \in C} prob(c)}$$

for all $C \subseteq \mathcal{I}_{\rm f}(P)$. ■

Definition 7. *Let $P \in \mathcal{P}_{\rm C,pc}$ and $c \in \mathcal{C}_{\rm f}(P)$. The average duration of c is the sequence of the average sojourn times in the states traversed by c, which is defined by induction on the length of c through the following $(\mathbb{R}_{>0})^*$-valued function:*

$$\boxed{time_{\rm a}(c) = \begin{cases} \varepsilon & \text{if } length(c) = 0 \\ \frac{1}{rate_{\rm t}(P,0)} \circ time_{\rm a}(c') & \text{if } c \equiv P \xrightarrow{a,\lambda} c' \end{cases}}$$

where ε is the empty average duration. We also define the multiset of the computations of C whose average duration is not greater than θ as:

$$C_{\leq\theta} = \{\!|\, c \in C \mid length(c) \leq length(\theta) \wedge \forall i = 1, \ldots, length(c).\, time_{\rm a}(c)[i] \leq \theta[i] \,|\!\}$$

for all $C \subseteq \mathcal{C}_{\rm f}(P)$ and $\theta \in (\mathbb{R}_{>0})^*$. ■

Definition 8. *Let $P \in \mathcal{P}_{\rm C,pc}$ and $c \in \mathcal{C}_{\rm f}(P)$. The duration of c is the sequence of the random variables quantifying the sojourn times in the states traversed by c, which is defined by induction on the length of c through the following random-variable-sequence-valued function:*

$$time_{\rm d}(c) = \begin{cases} \varepsilon & \text{if } length(c) = 0 \\ Exp_{rate_{\rm t}(P,0)} \circ time_{\rm d}(c') & \text{if } c \equiv P \xrightarrow{a,\lambda} c' \end{cases}$$

where ε is the empty duration while $Exp_{rate_{\rm t}(P,0)}$ is the exponentially distributed random variable with rate $rate_{\rm t}(P,0) \in \mathbb{R}_{>0}$. ■

Definition 9. *Let $P \in \mathcal{P}_{\rm C,pc}$, $C \subseteq \mathcal{I}_{\rm f}(P)$, and $\theta \in (\mathbb{R}_{>0})^*$. The probability distribution of executing a computation of C within a sequence θ of time units is given by:*

$$prob_{\rm d}(C,\theta) = \sum_{c \in C}^{length(c) \leq length(\theta)} prob(c) \cdot \prod_{i=1}^{length(c)} \Pr(time_{\rm d}(c)[i] \leq \theta[i])$$

where $\Pr(time_{\rm d}(c)[i] \leq \theta[i]) = 1 - {\rm e}^{-\theta[i]/time_{\rm a}(c)[i]}$ is the cumulative distribution function of the exponentially distributed random variable $time_{\rm d}(c)[i]$, whose expected value is $time_{\rm a}(c)[i]$. ■

We conclude by observing that the average duration (resp. duration) of a finite-length computation has been defined as the sequence of the average sojourn times (resp. of the random variables quantifying the sojourn times) in the states traversed by the computation. The same quantity could have been defined as the sum of the same basic ingredients, but this would not have been appropriate.

Example 1. Consider the two following process terms:

$$<g,\gamma>.<a,\lambda>.<b,\mu>.\underline{0} + <g,\gamma>.<a,\mu>.<d,\lambda>.\underline{0}$$
$$<g,\gamma>.<a,\lambda>.<d,\mu>.\underline{0} + <g,\gamma>.<a,\mu>.<b,\lambda>.\underline{0}$$

with $\lambda \neq \mu$ and $b \neq d$. Observed that the two terms have identical non-maximal computations, we further notice that the first term has the two following maximal computations each with probability 1/2:

$$c_{1,1} \equiv . \xrightarrow{g,\gamma} . \xrightarrow{a,\lambda} . \xrightarrow{b,\mu} .$$
$$c_{1,2} \equiv . \xrightarrow{g,\gamma} . \xrightarrow{a,\mu} . \xrightarrow{d,\lambda} .$$

while the second term has the two following maximal computations each with probability 1/2:

$$c_{2,1} \equiv . \xrightarrow{g,\gamma} . \xrightarrow{a,\lambda} . \xrightarrow{d,\mu} .$$
$$c_{2,2} \equiv . \xrightarrow{g,\gamma} . \xrightarrow{a,\mu} . \xrightarrow{b,\lambda} .$$

If the average duration were defined as the sum of the average sojourn times, then $c_{1,1}$ and $c_{2,2}$ would have the same trace $g \circ a \circ b$ and the same average duration $\frac{1}{2 \cdot \gamma} + \frac{1}{\lambda} + \frac{1}{\mu}$, and similarly $c_{1,2}$ and $c_{2,1}$ would have the same trace $g \circ a \circ d$ and the same average duration $\frac{1}{2 \cdot \gamma} + \frac{1}{\mu} + \frac{1}{\lambda}$. This would lead to conclude that the two terms are equivalent, whereas an external observer equipped with a button-pushing machine displaying the names of the actions that are performed and the times at which they are performed [49] would be able to distinguish between the two terms. ■

3 Markovian Bisimilarity

Markovian bisimilarity considers two process terms to be equivalent whenever they are able to mimic each other's functional and performance behavior stepwise. In this section we provide the definition of Markovian bisimilarity over $\mathcal{P}_{\mathrm{C}}$ and we recall its properties from [32,15,31,14,18,4,23].

3.1 Equivalence Definition

The basic idea behind Markovian bisimilarity is that, whenever a process term can perform actions with a certain name that reach a certain set of terms at a certain speed, then any process term equivalent to the given one has to be able to respond with actions with the same name that reach an equivalent set of terms at the same speed. This can be formalized through the comparison of the process term exit rates when executing actions of the same name (and level) that lead to the same set of equivalent terms.

Definition 10. *An equivalence relation $\mathcal{B} \subseteq \mathcal{P}_{\mathrm{C}} \times \mathcal{P}_{\mathrm{C}}$ is a Markovian bisimulation iff, whenever $(P_1, P_2) \in \mathcal{B}$, then for all action names $a \in Name$, levels $l \in \{0, -1\}$, and equivalence classes $C \in \mathcal{P}_{\mathrm{C}}/\mathcal{B}$:*

$$rate(P_1, a, l, C) = rate(P_2, a, l, C)$$ ■

Since the union of all the Markovian bisimulations can be proved to be the largest Markovian bisimulation, the definition below follows.

Definition 11. *Markovian bisimilarity, denoted by $\sim_{\mathrm{MB}}$, is the union of all the Markovian bisimulations.* ■

Obviously, $\sim_{\mathrm{MB}}$ is strictly finer than classical bisimilarity [43,41] and probabilistic bisimilarity [40]. We conclude with an easy-to-check necessary condition.

Proposition 1. *Let $P_1, P_2 \in \mathcal{P}_{\mathrm{C}}$. Whenever $P_1 \sim_{\mathrm{MB}} P_2$, then for all $a \in Name$ and $l \in \{0, -1\}$:*

$$rate(P_1, a, l, \mathcal{P}_{\mathrm{C}}) = rate(P_2, a, l, \mathcal{P}_{\mathrm{C}})$$ ■

3.2 Exactness

Markovian bisimilarity is consistent with an exact aggregation for CTMCs that is known under the name of ordinary lumping.

Definition 12. *A partition $\mathcal{O}$ of the state space of a CTMC is an ordinary lumping iff, whenever $s_1, s_2 \in O$ for some $O \in \mathcal{O}$, then for all $O' \in \mathcal{O}$:*
$\sum\{\!|\, \lambda \in \mathbf{R}_{>0} \mid \exists s' \in O'.\, s_1 \stackrel{\lambda}{\longrightarrow} s' \,|\!\} = \sum\{\!|\, \lambda \in \mathbf{R}_{>0} \mid \exists s' \in O'.\, s_2 \stackrel{\lambda}{\longrightarrow} s' \,|\!\}$ ■

Theorem 1. *The CTMC-level aggregation induced by $\sim_{\rm MB}$ is an ordinary lumping.* ■

Corollary 1. *The CTMC-level aggregation induced by $\sim_{\rm MB}$ is exact.* ■

3.3 Congruence

Markovian bisimilarity is a congruence with respect to all the operators of CMPC.

Theorem 2. *Let $P_1, P_2 \in \mathcal{P}_{\rm C}$. Whenever $P_1 \sim_{\rm MB} P_2$, then:*

1. *$<a, \tilde{\lambda}>.P_1 \sim_{\rm MB} <a, \tilde{\lambda}>.P_2$ for all $<a, \tilde{\lambda}> \in Act_{\rm C}$.*
2. *$P_1 + P \sim_{\rm MB} P_2 + P$ and $P + P_1 \sim_{\rm MB} P + P_2$ for all $P \in \mathcal{P}_{\rm C}$.*
3. *$P_1 \parallel_S P \sim_{\rm MB} P_2 \parallel_S P$ and $P \parallel_S P_1 \sim_{\rm MB} P \parallel_S P_2$ for all $S \subseteq Name$ and $P \in \mathcal{P}_{\rm C}$.* ■

As far as recursion is concerned, we need to extend $\sim_{\rm MB}$ to open process terms. These are terms containing free process variables, i.e. variables not occurring within the scope of a *rec* binder.

Definition 13. *Let $P_1, P_2 \in \mathcal{L}_{\rm C}$ containing a free process variable X. We define $P_1 \sim_{\rm MB} P_2$ iff $P_1\{Q/X\} \sim_{\rm MB} P_2\{Q/X\}$ for all $Q \in \mathcal{P}_{\rm C}$.* ■

Theorem 3. *Let $P_1, P_2 \in \mathcal{L}_{\rm C}$ containing a free process variable X. Whenever $P_1 \sim_{\rm MB} P_2$, then $rec\, X : P_1 \sim_{\rm MB} rec\, X : P_2$.* ■

3.4 Axiomatization

Markovian bisimilarity has a sound and complete axiomatization over $\mathcal{P}_{\rm C}$, whose specific axioms are shown Table 1. This includes three basic laws for alternative composition, two laws characterizing the race policy and the reactive preselection policy, an expansion law for parallel composition, and three laws for recursion. As far as $\mathcal{A}_6^{\rm MB}$ is concerned, I and J are finite index sets (if empty, the related summations are taken to be $\underline{0}$). The validity of this law is a consequence of the memoryless property of the exponentially distributed durations and of the fact that the probability that two concurrent exponentially timed actions terminate simultaneously is zero, which allows the semantics for the parallel composition operator to be defined in the usual interleaving style.

Theorem 4. *The deduction system $DED(\mathcal{A}^{\rm MB})$ is sound and complete for $\sim_{\rm MB}$ over $\mathcal{P}_{\rm C}$, i.e. for all $P_1, P_2 \in \mathcal{P}_{\rm C}$:*
$$P_1 \sim_{\rm MB} P_2 \iff \mathcal{A}^{\rm MB} \vdash P_1 = P_2$$ ■

Table 1. Axiomatization of $\sim_{\mathrm{MB}}$ over $\mathcal{P}_{\mathrm{C}}$

$$
\begin{array}{ll}
(\mathcal{A}_1^{\mathrm{MB}}) & P_1 + P_2 = P_2 + P_1 \\
(\mathcal{A}_2^{\mathrm{MB}}) & (P_1 + P_2) + P_3 = P_1 + (P_2 + P_3) \\
(\mathcal{A}_3^{\mathrm{MB}}) & P + \underline{0} = P \\
(\mathcal{A}_4^{\mathrm{MB}}) & <a, \lambda_1>.P + <a, \lambda_2>.P = <a, \lambda_1 + \lambda_2>.P \\
(\mathcal{A}_5^{\mathrm{MB}}) & <a, *_{w_1}>.P + <a, *_{w_2}>.P = <a, *_{w_1 + w_2}>.P \\
(\mathcal{A}_6^{\mathrm{MB}}) & \sum_{i \in I} <a_i, \tilde{\lambda}_i>.P_{1,i} \parallel_S \sum_{j \in J} <b_j, \tilde{\mu}_j>.P_{2,j} = \\
& \quad \sum_{k \in I, a_k \notin S} <a_k, \tilde{\lambda}_k>.\left(P_{1,k} \parallel_S \sum_{j \in J} <b_j, \tilde{\mu}_j>.P_{2,j} \right) + \\
& \quad \sum_{h \in J, b_h \notin S} <b_h, \tilde{\mu}_h>.\left(\sum_{i \in I} <a_i, \tilde{\lambda}_i>.P_{1,i} \parallel_S P_{2,h} \right) + \\
& \quad \sum_{k \in I, a_k \in S, \tilde{\lambda}_k \in \mathbb{R}_{>0}} \; \sum_{h \in J, b_h = a_k, \tilde{\mu}_h = *_{w_h}} <a_k, \tilde{\lambda}_k \cdot \frac{w_h}{weight(P_2, b_h)}>.(P_{1,k} \parallel_S P_{2,h}) + \\
& \quad \sum_{h \in J, b_h \in S, \tilde{\mu}_h \in \mathbb{R}_{>0}} \; \sum_{k \in I, a_k = b_h, \tilde{\lambda}_k = *_{v_k}} <b_h, \tilde{\mu}_h \cdot \frac{v_k}{weight(P_1, a_k)}>.(P_{1,k} \parallel_S P_{2,h}) + \\
& \quad \sum_{k \in I, a_k \in S, \tilde{\lambda}_k = *_{v_k}} \; \sum_{h \in J, b_h = a_k, \tilde{\mu}_h = *_{w_h}} \\
& \qquad <a_k, *_{\frac{v_k}{weight(P_1, a_k)} \cdot \frac{w_h}{weight(P_2, b_h)} \cdot (weight(P_1, a_k) + weight(P_2, b_h))}>.(P_{1,k} \parallel_S P_{2,h}) \\
(\mathcal{A}_7^{\mathrm{MB}}) & rec\ X : P = rec\ Y : P\{Y/X\} \qquad \text{if } Y \text{ not in } P \\
(\mathcal{A}_8^{\mathrm{MB}}) & rec\ X : P = P\{(rec\ X : P)/X\} \\
(\mathcal{A}_9^{\mathrm{MB}}) & Q = P\{Q/X\} \Rightarrow Q = rec\ X : P
\end{array}
$$

3.5 Logical Characterization

Markovian bisimilarity has a modal logic characterization over $\mathcal{P}_{\mathrm{C,pc}}$ based on a Markovian variant of the Hennessy-Milner logic [29], in which the diamond operator is decorated with a rate lower bound.

Definition 14. *The set of the formulas of* $\mathrm{HML}_{\mathrm{MB}}$ *is generated by the following syntax:*

$$\phi ::= \mathrm{true} \mid \neg\phi \mid \phi \wedge \phi \mid \nabla_a \mid \langle a \rangle_\lambda \phi$$

where $a \in Name$ *and* $\lambda \in \mathbb{R}_{>0}$. ■

Definition 15. *The satisfaction relation* $\models_{\mathrm{MB}}$ *of* $\mathrm{HML}_{\mathrm{MB}}$ *over* $\mathcal{P}_{\mathrm{C,pc}}$ *is defined by structural induction as follows:*

$$
\begin{array}{ll}
P \models_{\mathrm{MB}} \mathrm{true} & \\
P \models_{\mathrm{MB}} \neg\phi & \text{if } P \not\models_{\mathrm{MB}} \phi \\
P \models_{\mathrm{MB}} \phi_1 \wedge \phi_2 & \text{if } P \models_{\mathrm{MB}} \phi_1 \text{ and } P \models_{\mathrm{MB}} \phi_2 \\
P \models_{\mathrm{MB}} \nabla_a & \text{if } rate(P, a, 0, \mathcal{P}_{\mathrm{C}}) = 0 \\
P \models_{\mathrm{MB}} \langle a \rangle_\lambda \phi & \text{if } rate(P, a, 0, sat(\phi)) \geq \lambda
\end{array}
$$

where $sat(\phi) = \{P' \in \mathcal{P}_{\mathrm{C}} \mid P' \models_{\mathrm{MB}} \phi\}$. ■

Theorem 5. *Let $P_1, P_2 \in \mathcal{P}_{\mathrm{C,pc}}$. Then:*

$$P_1 \sim_{\mathrm{MB}} P_2 \iff (\forall \phi \in \mathrm{HML}_{\mathrm{MB}}.\, P_1 \models_{\mathrm{MB}} \phi \iff P_2 \models_{\mathrm{MB}} \phi) \qquad \blacksquare$$

We also mention that Markovian bisimilarity has a temporal logic characterization based on the branching-time logic CSL [3]. Besides the usual propositional connectives, this logic comprises:

- a probability operator, which replaces the universal and existential computation quantifiers and allows to refer to the probability of performing a computation that satisfies a certain formula;
- a time-bounded next operator, which expresses that the next state is reached within a certain amount of time and a certain formula holds in it;
- a time-bounded until operator, which expresses that a certain formula is satisfied at some instant no later than a certain amount of time and at all preceding instants another given formula holds;
- a steady-state operator, which enables to reason about the probability to be in the long run in some state that satisfies a certain formula.

3.6 Verification Complexity

Markovian bisimilarity can be decided in polynomial time by means of a partition refinement algorithm in the style of [42]. This algorithm exploits the fact that $\sim_{\mathrm{MB}}$ can be characterized as a fixed point of successively finer relations. In fact we have:

$$\sim_{\mathrm{MB}} = \bigcap_{i \in \mathbb{N}} \sim_{\mathrm{MB},i}$$

where $\sim_{\mathrm{MB},0} = \mathcal{P}_{\mathrm{C}} \times \mathcal{P}_{\mathrm{C}}$ and $\sim_{\mathrm{MB},i}$ for $i \geq 1$ is defined as follows: whenever $(P_1, P_2) \in \sim_{\mathrm{MB},i}$, then for all $a \in \mathit{Name}$, $l \in \{0, -1\}$, and $C \in \mathcal{P}_{\mathrm{C}}/\sim_{\mathrm{MB},i-1}$:

$$\mathit{rate}(P_1, a, l, C) = \mathit{rate}(P_2, a, l, C)$$

In other words, $\sim_{\mathrm{MB},0}$ induces a trivial partition with a single equivalence class that coincides with $\mathcal{P}_{\mathrm{C}}$, $\sim_{\mathrm{MB},1}$ refines the previous partition by creating an equivalence class for each set of terms that satisfy the necessary condition stated by Prop. 1, and so on.

The algorithm to check whether $P_1 \sim_{\mathrm{MB}} P_2$ thus proceeds as follows:

1. Build a partition with a single class including all the states of the disjoint union of $[\![P_1]\!]$ and $[\![P_2]\!]$, then initialize a list of splitters with this class.
2. Refine the current partition by splitting each of its classes according to the exit rates towards one of the splitters, then remove this splitter from the list.
3. For each split class, insert into the list of splitters all the resulting subclasses except for the largest one.
4. If the list of splitters is empty, return yes/no depending on whether the initial state of $[\![P_1]\!]$ and the initial state of $[\![P_2]\!]$ belong to the same class or not, otherwise go back to the refinement step.

The time complexity is $O(m \cdot \log n)$, where n is the number of states and m is the number of transitions of the disjoint union of $[\![P_1]\!]$ and $[\![P_2]\!]$. To achieve this complexity it is necessary to resort to a splay tree when representing the subclasses arising from the splitting of a class.

4 Variants of Markovian Bisimilarity

Due to the nice properties presented in the previous section, Markovian bisimilarity has received more attention than Markovian testing and trace equivalences. For this reason, some useful variants of it have appeared in the literature, three of which will be recalled in this section.

The first one – reward Markovian bisimilarity [10] – takes rewards into account in order to allow process terms to be compositionally manipulated in a way that is sensitive to specific performance measures. The second one – interactive Markovian bisimilarity [30] – stems from the interaction of nondeterministic process calculi and CTMCs and deals with both Markovian branchings and nondeterministic branchings. The third one – extended Markovian bisimilarity [8] – considers immediate actions à la GSPN [1] and deals with both Markovian branchings and prioritized/probabilistic branchings.

4.1 Reward Markovian Bisimilarity

Although Markovian bisimilarity is consistent with ordinary lumping, which is an exact aggregation, it may happen that specific performance measures distinguish between ordinarily lumpable states by ascribing them a different meaning.

The usual approach to the specification of performance measures for CTMC-based models relies on reward structures [34]. This requires associating real numbers with model behaviors and activities, which are then transferred to the proper states (as yield rewards) and transitions (as bonus rewards) of the underlying CTMC, respectively. A yield reward expresses the rate at which a gain (or a loss, if the number is negative) is accumulated while sojourning in the related state. By contrast, a bonus reward specifies the instantaneous gain (or loss) implied by the execution of the related transition.

The instant-of-time value of a performance measure specified through a reward structure is computed as follows for a CTMC:

$$\boxed{\sum_{s} yr(s) \cdot \pi(s) + \sum_{s \xrightarrow{\lambda} s'} br(s, \lambda, s') \cdot \phi(s, \lambda, s')}$$

where:

- $yr(s)$ is the yield reward associated with state s.
- $\pi(s)$ is the probability of being in state s at the considered instant of time.
- $br(s, \lambda, s')$ is the bonus reward associated with transition $s \xrightarrow{\lambda} s'$.
- $\phi(s, \lambda, s')$ is the frequency of transition $s \xrightarrow{\lambda} s'$ at the considered instant of time, which is given by $\pi(s) \cdot \lambda$.

In this setting, ascribing a different meaning to ordinarily lumpable states amounts to giving different rewards to such states or their outgoing transitions. In order to manipulate process terms in a performance-sensitive way, the definition of Markovian bisimilarity can be modified by taking rewards into account.

Before doing that, we need to extend CMPC with rewards. While bonus rewards can naturally be associated with actions, the treatment of yield rewards is not trivial because in process calculi the concept of state is implicit. An approach that can be followed is to associate yield rewards with actions too, with the yield reward of a state being given by the sum of the yield rewards associated with the actions enabled in that state (additivity assumption).

Therefore, we derive from CMPC a new calculus, which we call concurrent reward Markovian process calculus (CRMPC):

– The syntax is extended as follows:

$$\begin{array}{l} P ::= \underline{0} \\ \quad | \quad <a, \lambda, yr, br>.P \\ \quad | \quad <a, *_w, *, *>.P \\ \quad | \quad P + P \\ \quad | \quad P \parallel_S P \\ \quad | \quad X \\ \quad | \quad rec\, X : P \end{array}$$

where $yr \in \mathbb{R}$ is the yield reward and $br \in \mathbb{R}$ is the bonus reward. We denote by $\mathcal{P}_{\mathrm{CR}}$ the set of the closed and guarded process terms of CRMPC.

– The semantic rules are modified accordingly. In particular, the synchronization rules change as follows:

$$\frac{P_1 \xrightarrow{a,\lambda,yr,br} P_1' \qquad P_2 \xrightarrow{a,*_w,*,*} P_2' \qquad a \in S}{P_1 \parallel_S P_2 \xrightarrow{a,\lambda \cdot \frac{w}{weight(P_2,a)}, yr \cdot \frac{w}{weight(P_2,a)}, br} P_1' \parallel_S P_2'}$$

$$\frac{P_1 \xrightarrow{a,*_w,*,*} P_1' \qquad P_2 \xrightarrow{a,\lambda,yr,br} P_2' \qquad a \in S}{P_1 \parallel_S P_2 \xrightarrow{a,\lambda \cdot \frac{w}{weight(P_1,a)}, yr \cdot \frac{w}{weight(P_1,a)}, br} P_1' \parallel_S P_2'}$$

$$\frac{P_1 \xrightarrow{a,*_{w_1},*,*} P_1' \qquad P_2 \xrightarrow{a,*_{w_2},*,*} P_2' \qquad a \in S}{P_1 \parallel_S P_2 \xrightarrow{a,* \frac{w_1}{weight(P_1,a)} \cdot \frac{w_2}{weight(P_2,a)} \cdot (weight(P_1,a) + weight(P_2,a)),*,*} P_1' \parallel_S P_2'}$$

Note that the yield rewards are subject to the same normalization as the rates, because they are strictly related to the sojourn time in the states. By contrast, no normalization is needed for the bonus rewards, as they are earned upon the execution of the transitions.

We now introduce the concept of exit reward, on the basis of which we shall define the performance-sensitive variant of Markovian bisimilarity.

Definition 16. *Let $P \in \mathcal{P}_{\mathrm{CR}}$, $a \in Name$, $l \in \{0, -1\}$, and $C \subseteq \mathcal{P}_{\mathrm{CR}}$. The exit reward of P when executing actions of name a and level l that lead to C is defined through the following real function:*

$$reward(P,a,l,C) = \begin{cases} \sum\{\!|\, yr+\lambda\cdot br \in \mathbb{R} \mid \exists P' \in C.\, P \xrightarrow{a,\lambda,yr,br} P' \,|\!\} & \text{if } l = 0 \\ 0 & \text{if } l = -1 \end{cases}$$

where the summation is taken to be zero whenever its multiset is empty. ■

Definition 17. *An equivalence relation $\mathcal{B} \subseteq \mathcal{P}_{\rm CR} \times \mathcal{P}_{\rm CR}$ is a reward Markovian bisimulation iff, whenever $(P_1,P_2) \in \mathcal{B}$, then for all action names $a \in Name$, levels $l \in \{0,-1\}$, and equivalence classes $C \in \mathcal{P}_{\rm CR}/\mathcal{B}$:*

$$rate(P_1,a,l,C) = rate(P_2,a,l,C)$$
$$reward(P_1,a,l,C) = reward(P_2,a,l,C)$$ ■

Definition 18. *Reward Markovian bisimilarity, denoted by $\sim_{\rm RMB}$, is the union of all the reward Markovian bisimulations.* ■

$\sim_{\rm RMB}$ enjoys the same properties as $\sim_{\rm MB}$. The characterizing axioms are:

$$\begin{array}{c} <a,\lambda_1,yr_1,br_1>.P + <a,\lambda_2,yr_2,br_2>.P = \\ <a,\lambda_1+\lambda_2,yr_1+yr_2,\frac{\lambda_1}{\lambda_1+\lambda_2}\cdot br_1 + \frac{\lambda_2}{\lambda_1+\lambda_2}\cdot br_2>.P \\ <a,*_{w_1},*,*>.P + <a,*_{w_2},*,*>.P = <a,*_{w_1+w_2},*,*>.P \end{array}$$

or equivalently:

$$\begin{array}{c} <a,\lambda,yr,br>.P = <a,\lambda,yr+\lambda\cdot br,0>.P \\ <a,\lambda_1,yr_1,0>.P + <a,\lambda_2,yr_2,0>.P = <a,\lambda_1+\lambda_2,yr_1+yr_2,0>.P \\ <a,*_{w_1},*,*>.P + <a,*_{w_2},*,*>.P = <a,*_{w_1+w_2},*,*>.P \end{array}$$

or equivalently:

$$\begin{array}{c} <a,\lambda,yr,br>.P = <a,\lambda,0,br+\frac{yr}{\lambda}>.P \\ <a,\lambda_1,0,br_1>.P + <a,\lambda_2,0,br_2>.P = <a,\lambda_1+\lambda_2,0,\frac{\lambda_1}{\lambda_1+\lambda_2}\cdot br_1 + \frac{\lambda_2}{\lambda_1+\lambda_2}\cdot br_2>.P \\ <a,*_{w_1},*,*>.P + <a,*_{w_2},*,*>.P = <a,*_{w_1+w_2},*,*>.P \end{array}$$

4.2 Interactive Markovian Bisimilarity

Markovian bisimilarity deals with fully probabilistic models. However, it may happen that not all the details of a model are known in the early design stages. In that case, nondeterminism is a useful abstraction.

A way to recover nondeterminism in a Markovian framework is to combine nondeterministic process calculi and CTMCs. This can be accomplished by replacing durational actions with two distinct prefix operators – one for interacting actions that take no time and one for exponential delays – with the inter-process communication being implemented through the synchronization of visible interacting actions. Since an invisible action – which we denote by τ as usual – takes no time and cannot be prevented by any synchronization constraint, we assume maximal progress, i.e. τ-actions take precedence over time passing.

We now derive from CMPC a new calculus, which we call concurrent interactive Markovian process calculus (CIMPC):

– The syntax is modified as follows:

$$\boxed{\begin{array}{l} P ::= \underline{0} \\ \quad \mid\ a.P \\ \quad \mid\ (\lambda).P \\ \quad \mid\ P + P \\ \quad \mid\ P \parallel_S P \\ \quad \mid\ X \\ \quad \mid\ rec\, X : P \end{array}}$$

where $a \in Name \cup \{\tau\}$. We denote by $\mathcal{P}_{\rm CI}$ the set of the closed and guarded process terms of CIMPC.

– Two transition relations are necessary: one for interacting actions and one for exponential delays. Besides handling recursion, the semantic rules for interacting actions include:

$$\boxed{\begin{array}{c} a.P \stackrel{a}{\longrightarrow}_{\rm I} P \\[2ex] \dfrac{P_1 \stackrel{a}{\longrightarrow}_{\rm I} P'}{P_1 + P_2 \stackrel{a}{\longrightarrow}_{\rm I} P'} \qquad \dfrac{P_2 \stackrel{a}{\longrightarrow}_{\rm I} P'}{P_1 + P_2 \stackrel{a}{\longrightarrow}_{\rm I} P'} \\[2ex] \dfrac{P_1 \stackrel{a}{\longrightarrow}_{\rm I} P_1' \quad a \notin S}{P_1 \parallel_S P_2 \stackrel{a}{\longrightarrow}_{\rm I} P_1' \parallel_S P_2} \qquad \dfrac{P_2 \stackrel{a}{\longrightarrow}_{\rm I} P_2' \quad a \notin S}{P_1 \parallel_S P_2 \stackrel{a}{\longrightarrow}_{\rm I} P_1 \parallel_S P_2'} \\[2ex] \dfrac{P_1 \stackrel{a}{\longrightarrow}_{\rm I} P_1' \quad P_2 \stackrel{a}{\longrightarrow}_{\rm I} P_2' \quad a \in S}{P_1 \parallel_S P_2 \stackrel{a}{\longrightarrow}_{\rm I} P_1' \parallel_S P_2'} \end{array}}$$

while the semantic rules for exponential delays include:

$$\boxed{\begin{array}{c} (\lambda).P \stackrel{\lambda}{\longrightarrow}_{\rm M} P \\[2ex] \dfrac{P_1 \stackrel{\lambda}{\longrightarrow}_{\rm M} P'}{P_1 + P_2 \stackrel{\lambda}{\longrightarrow}_{\rm M} P'} \qquad \dfrac{P_2 \stackrel{\lambda}{\longrightarrow}_{\rm M} P'}{P_1 + P_2 \stackrel{\lambda}{\longrightarrow}_{\rm M} P'} \\[2ex] \dfrac{P_1 \stackrel{\lambda}{\longrightarrow}_{\rm M} P_1'}{P_1 \parallel_S P_2 \stackrel{\lambda}{\longrightarrow}_{\rm M} P_1' \parallel_S P_2} \qquad \dfrac{P_2 \stackrel{\lambda}{\longrightarrow}_{\rm M} P_2'}{P_1 \parallel_S P_2 \stackrel{\lambda}{\longrightarrow}_{\rm M} P_1 \parallel_S P_2'} \end{array}}$$

Before defining the interactive variant of Markovian bisimilarity, we adapt to this setting the concept of exit rate.

Definition 19. *Let $P \in \mathcal{P}_{\rm CI}$ and $C \subseteq \mathcal{P}_{\rm CI}$. The exit rate of P towards C is defined through the following non-negative real function:*

$$\boxed{rate_{\rm M}(P, C) = \sum\{\!|\, \lambda \in \mathbb{R}_{>0} \mid \exists P' \in C.\, P \stackrel{\lambda}{\longrightarrow}_{\rm M} P' \,|\!\}}$$

where the summation is taken to be zero whenever its multiset is empty. ■

Definition 20. *An equivalence relation $\mathcal{B} \subseteq \mathcal{P}_{\rm CI} \times \mathcal{P}_{\rm CI}$ is an interactive Markovian bisimulation iff, whenever $(P_1, P_2) \in \mathcal{B}$, then for all action names $a \in Name \cup \{\tau\}$ and equivalence classes $C \in \mathcal{P}_{\rm CI}/\mathcal{B}$:*

- $P_1 \stackrel{a}{\longrightarrow}_{\rm I} P_1'$ *implies* $P_2 \stackrel{a}{\longrightarrow}_{\rm I} P_2'$ *for some* P_2' *with* $(P_1', P_2') \in \mathcal{B}$.
- $P_1 \stackrel{\tau}{\not\longrightarrow}_{\rm I}$ *implies* $rate_{\rm M}(P_1, C) = rate_{\rm M}(P_2, C)$. ■

Definition 21. *Interactive Markovian bisimilarity, denoted by $\sim_{\rm IMB}$, is the union of all the interactive Markovian bisimulations.* ■

$\sim_{\rm IMB}$ enjoys properties similar to those of $\sim_{\rm MB}$. The characterizing axioms are:

$$\boxed{\begin{array}{c} a.P + a.P = a.P \\ (\lambda_1).P + (\lambda_2).P = (\lambda_1 + \lambda_2).P \\ \tau.P + (\lambda).Q = \tau.P \end{array}}$$

which encode idempotency, race policy, and maximal progress, respectively.

Since τ-actions are invisible and take no time, when comparing process terms they should be abstracted away. This can be achieved by weakening $\sim_{\rm IMB}$ in such a way that, after any non-pre-emptable exponential delay, all the states that can evolve via a finite sequence of τ-transitions to a given class are skipped.

Definition 22. *Let $C \subseteq \mathcal{P}_{\rm CI}$. The internal backward closure of C is defined as follows:*

$$C_\tau = \{P' \in \mathcal{P}_{\rm CI} \mid \exists P \in C.\, P' \stackrel{\tau^*}{\Longrightarrow}_{\rm I} P\}$$

where $P' \stackrel{\tau^*}{\Longrightarrow}_{\rm I} P$ means that P' can evolve to P after a finite sequence of zero or more τ-transitions. ■

Definition 23. *An equivalence relation $\mathcal{B} \subseteq \mathcal{P}_{\rm CI} \times \mathcal{P}_{\rm CI}$ is a weak interactive Markovian bisimulation iff, whenever $(P_1, P_2) \in \mathcal{B}$, then for all action names $a \in Name$ and equivalence classes $C \in \mathcal{P}_{\rm CI}/\mathcal{B}$:*

- $P_1 \stackrel{a}{\longrightarrow}_{\rm I} P_1'$ *implies* $P_2 \stackrel{\tau^* a \tau^*}{\Longrightarrow}_{\rm I} P_2'$ *for some* P_2' *with* $(P_1', P_2') \in \mathcal{B}$.
- $P_1 \stackrel{\tau}{\longrightarrow}_{\rm I} P_1'$ *implies* $P_2 \stackrel{\tau^*}{\Longrightarrow}_{\rm I} P_2'$ *for some* P_2' *with* $(P_1', P_2') \in \mathcal{B}$.
- $P_1 \stackrel{\tau^*}{\Longrightarrow}_{\rm I} P_1' \stackrel{\tau}{\not\longrightarrow}_{\rm I}$ *implies* $P_2 \stackrel{\tau^*}{\Longrightarrow}_{\rm I} P_2' \stackrel{\tau}{\not\longrightarrow}_{\rm I}$ *for some* P_2' *with* $rate_{\rm M}(P_1', C_\tau) = rate_{\rm M}(P_2', C_\tau)$. ■

Definition 24. *Weak interactive Markovian bisimilarity, denoted by $\approx_{\rm IMB}$, is the union of all the weak interactive Markovian bisimulations.* ■

$\approx_{\rm IMB}$ is strictly coarser than $\sim_{\rm IMB}$ but it is not a congruence with respect to alternative composition. As usual, initial τ-actions need a different treatment.

Definition 25. *Let $P_1, P_2 \in \mathcal{P}_{\rm CI}$. We say that P_1 is weakly interactive Markovian bisimulation congruent to P_2, written $P_1 \simeq_{\rm IMB} P_2$, iff for all action names $a \in Name \cup \{\tau\}$ and equivalence classes $C \in \mathcal{P}_{\rm CI}/\approx_{\rm IMB}$:*

- $P_1 \stackrel{a}{\longrightarrow}_{\rm I} P_1'$ *implies* $P_2 \stackrel{\tau^* a \tau^*}{\Longrightarrow}_{\rm I} P_2'$ *for some* P_2' *with* $P_1' \approx_{\rm IMB} P_2'$.
- $P_2 \stackrel{a}{\longrightarrow}_{\rm I} P_2'$ *implies* $P_1 \stackrel{\tau^* a \tau^*}{\Longrightarrow}_{\rm I} P_1'$ *for some* P_1' *with* $P_1' \approx_{\rm IMB} P_2'$.
- $P_1 \not\stackrel{\tau}{\longrightarrow}_{\rm I}$ *iff* $P_2 \not\stackrel{\tau}{\longrightarrow}_{\rm I}$.
- $P_1 \not\stackrel{\tau}{\longrightarrow}_{\rm I}$ *implies* $rate_{\rm M}(P_1, C) = rate_{\rm M}(P_2, C)$. ■

It turns out $\sim_{\rm IMB} \subset \simeq_{\rm IMB} \subset \approx_{\rm IMB}$ with $\simeq_{\rm IMB}$ enjoying the same properties as $\sim_{\rm IMB}$. Moreover, $\simeq_{\rm IMB}$ has the following additional characterizing axioms:

$$\begin{array}{c} a.\tau.P = a.P \\ P + \tau.P = \tau.P \\ a.(P + \tau.Q) + \tau.Q = a.(P + \tau.Q) \\ (\lambda).\tau.P = (\lambda).P \end{array}$$

which are called τ-laws and witness the capability of $\simeq_{\rm IMB}$ of abstracting from τ-actions.

4.3 Extended Markovian Bisimilarity

Markovian bisimilarity is restricted to exponential distributions. Although their combinations can approximate most of general distributions arbitrarily closely, some useful distributions are left out, specially the one representing a zero duration. The capability of expressing zero durations would also constitute a good performance abstraction mechanism, similarly to the functional abstraction mechanism given by the invisible action name τ.

In the modeling process it often happens to deal with choices among logical events (like the reception of a message vs. its loss) with which no timing can reasonably be associated, or to encounter activities that are several orders of magnitude faster than the activities that are important for the evaluation of certain performance measures. In all of these cases, using a zero duration would be an adequate solution from the modeling standpoint.

Zero durations can be introduced by admitting the so-called immediate actions. Each of them has a name and a zero duration (or, equivalently, an infinite rate), together with a priority level $l \in \mathbb{N}_{>0}$ and a weight $w \in \mathbb{R}_{>0}$. Priority levels and weights are used to choose among several immediate actions that are simultaneously enabled. According to the generative [26] preselection policy, each of the highest priority immediate actions that are enabled is given an execution probability proportional to its weight.

It is worth noting that this extended Markovian framework is complementary with respect to the interactive Markovian framework, because it is as if nondeterminism were ruled out by augmenting each interacting action with a priority level and a weight. Here maximal progress is subsumed by pre-emption: immediate τ-actions take precedence over all the lower priority actions.

We now derive from CMPC a new calculus, which we call concurrent extended Markovian process calculus (CEMPC):

– The syntax is extended as follows:

$$\begin{array}{l} P ::= \underline{0} \\ \quad \mid \; <a, \lambda>.P \\ \quad \mid \; <a, \infty_{l,w}>.P \\ \quad \mid \; <a, *_w^{l'}>.P \\ \quad \mid \; P + P \\ \quad \mid \; P \parallel_S P \\ \quad \mid \; X \\ \quad \mid \; rec\, X : P \end{array}$$

where $a \in Name \cup \{\tau\}$, $l \in \mathbf{N}_{>0}$, and $l' \in \mathbf{N}$. We denote by $\mathcal{P}_{\mathrm{CE}}$ the set of the closed and guarded process terms of CEMPC. We point out that every passive action has been augmented with a priority constraint, which is useful to keep under control the process priority interrelation. Besides synchronizing with passive actions with the same priority constraint, a passive action with priority constraint 0 can only synchronize with an exponentially timed action, while a passive action with priority constraint $l' > 0$ can only synchronize with an immediate action with priority level $l = l'$.

– The additional semantic rules specific for immediate actions are the following:

$$<a, \infty_{l,w}>.P \xrightarrow{a,\infty_{l,w}} P$$

$$\frac{P_1 \xrightarrow{a,\infty_{l,w}} P_1' \qquad P_2 \xrightarrow{a,*_v^l} P_2' \qquad a \in S}{P_1 \parallel_S P_2 \xrightarrow{a,\infty_{l,w} \cdot \frac{v}{weight(P_2,a,l)}} P_1' \parallel_S P_2'}$$

$$\frac{P_1 \xrightarrow{a,*_v^l} P_1' \qquad P_2 \xrightarrow{a,\infty_{l,w}} P_2' \qquad a \in S}{P_1 \parallel_S P_2 \xrightarrow{a,\infty_{l,w} \cdot \frac{v}{weight(P_1,a,l)}} P_1' \parallel_S P_2'}$$

Note that, consistently with the asymmetric action synchronization discipline, immediate actions can synchronize only with passive actions.

Before defining the extended variant of Markovian bisimilarity, we extend to immediate actions the notion of exit rate. Below, the priority level of an action is encoded through a number in $\mathbb{Z}$, which is 0 if the action is exponentially timed, l if the action rate is $\infty_{l,w}$, $-l-1$ if the action rate is $*_w^l$.

Definition 26. *Let $P \in \mathcal{P}_{\mathrm{CE}}$, $a \in Name \cup \{\tau\}$, $l \in \mathbb{Z}$, and $C \subseteq \mathcal{P}_{\mathrm{CE}}$. The exit rate of P when executing actions of name a and priority level l that lead to C is defined through the following non-negative real function:*

$$rate(P,a,l,C) = \begin{cases} \sum\{\!|\, \lambda \in \mathbf{R}_{>0} \mid \exists P' \in C.\, P \xrightarrow{a,\lambda} P' \,|\!\} & \text{if } l = 0 \\ \sum\{\!|\, w \in \mathbf{R}_{>0} \mid \exists P' \in C.\, P \xrightarrow{a,\infty_{l,w}} P' \,|\!\} & \text{if } l > 0 \\ \sum\{\!|\, w \in \mathbf{R}_{>0} \mid \exists P' \in C.\, P \xrightarrow{a,*_w^{-l-1}} P' \,|\!\} & \text{if } l < 0 \end{cases}$$

where each summation is taken to be zero whenever its multiset is empty. ■

In the following we denote by $pri^{\tau}_{\infty}(P)$ the priority level of the highest priority immediate τ-action enabled by P, and we set $pri^{\tau}_{\infty}(P) = 0$ if P does not enable any immediate τ-action. Moreover, given $l \in \mathbb{Z}$, we use *no-pre*(l, P) to denote that no action whose priority level is l can be pre-empted in P. Formally, this is the case whenever $l \geq pri^{\tau}_{\infty}(P)$ or $-l - 1 \geq pri^{\tau}_{\infty}(P)$.

Definition 27. *An equivalence relation $\mathcal{B} \subseteq \mathcal{P}_{\rm CE} \times \mathcal{P}_{\rm CE}$ is an extended Markovian bisimulation iff, whenever $(P_1, P_2) \in \mathcal{B}$, then for all action names $a \in Name \cup \{\tau\}$, equivalence classes $C \in \mathcal{P}_{\rm CE}/\mathcal{B}$, and priority levels $l \in \mathbb{Z}$ such that no-pre(l, P_1) and no-pre(l, P_2):*

$$rate(P_1, a, l, C) = rate(P_2, a, l, C)$$

■

Definition 28. *Extended Markovian bisimilarity, denoted by $\sim_{\rm EMB}$, is the union of all the extended Markovian bisimulations.* ■

$\sim_{\rm EMB}$ enjoys the same properties as $\sim_{\rm MB}$. The characterizing axioms are:

$$\begin{array}{|rcll|}
\hline
<a,\lambda_1>.P + <a,\lambda_2>.P &=& <a,\lambda_1+\lambda_2>.P & \\
<a,\infty_{l,w_1}>.P + <a,\infty_{l,w_2}>.P &=& <a,\infty_{l,w_1+w_2}>.P & \\
<a,*^{l}_{w_1}>.P + <a,*^{l}_{w_2}>.P &=& <a,*^{l}_{w_1+w_2}>.P & \\
<\tau,\infty_{l,w}>.P + <a,\lambda>.Q &=& <\tau,\infty_{l,w}>.P & \\
<\tau,\infty_{l,w}>.P + <a,\infty_{l',w'}>.Q &=& <\tau,\infty_{l,w}>.P & \text{if } l > l' \\
<\tau,\infty_{l,w}>.P + <a,*^{l'}_{w'}>.Q &=& <\tau,\infty_{l,w}>.P & \text{if } l > l' \\
\hline
\end{array}$$

with the last three encoding pre-emption.

Similarly to the interactive framework, when comparing process terms the immediate τ-actions should be abstracted away, as they are unobservable both from the functional viewpoint and from the performance viewpoint. However, weakening $\sim_{\rm EMB}$ is harder than weakening $\sim_{\rm IMB}$, because it is necessary to keep track of the priority levels and of the weights associated with the actions to be abstracted away. Furthemore, it is also necessary to take into account the degree of observability of the states.

Definition 29. *Let $P \in \mathcal{P}_{\rm CE}$ and $l \in \mathbb{N}_{>0}$. We say that P is l-unobservable iff $pri^{\tau}_{\infty}(P) = l$ and P does not enable any visible action with priority level $l' \in \mathbb{Z}$ such that $l' \geq l$ or $-l' - 1 \geq l$.* ■

Definition 30. *Let $n \in \mathbb{N}_{>0}$ and $P_1, P_2, \ldots, P_{n+1} \in \mathcal{P}_{\rm CE}$. A computation c of length n:*

$$P_1 \stackrel{\tau,\infty_{l_1,w_1}}{\longrightarrow} P_2 \stackrel{\tau,\infty_{l_2,w_2}}{\longrightarrow} \ldots \stackrel{\tau,\infty_{l_n,w_n}}{\longrightarrow} P_{n+1}$$

is unobservable iff for all $i = 1, \ldots, n$ process term P_i is l_i-unobservable. In that case, the probability of executing c is given by:

$$prob(c) = \prod_{i=1}^{n} \frac{w_i}{rate(P_i, \tau, l_i, \mathcal{P}_{\rm CE})}$$

■

Definition 31. *Let $P \in \mathcal{P}_{\mathrm{CE}}$, $a \in Name \cup \{\tau\}$, $l \in \mathbb{Z}$, and $C \subseteq \mathcal{P}_{\mathrm{CE}}$. The weak exit rate of P when executing actions with name a and priority level l that lead to C is defined through the following non-negative real function:*

$$\boxed{rate_{\mathrm{w}}(P, a, l, C) = \sum_{P' \in C_{\mathrm{w}}} rate(P, a, l, \{P'\}) \cdot prob_{\mathrm{w}}(P', C)}$$

where:

- *C_{w} is the weak backward closure of C:*
  $C_{\mathrm{w}} = C \cup \{Q \in \mathcal{P}_{\mathrm{CE}} - C \mid Q \text{ can reach } C \text{ via unobservable computations}\}$
- *$prob_{\mathrm{w}}$ is a $\mathbf{R}_{]0,1]}$-valued function representing the sum of the probabilities of all the unobservable computations from a term in C_{w} to C:*
  $$prob_{\mathrm{w}}(P', C) = \begin{cases} 1 & \text{if } P' \in C \\ \sum\{\!|\, prob(c) \mid c \text{ unobservable computation from } P' \text{ to } C \,|\!\} & \text{if } P' \in C_{\mathrm{w}} - C \end{cases}$$ ■

The definition of $\sim_{\mathrm{EMB}}$ can be weakened by using $rate_{\mathrm{w}}$ instead of $rate$ and by skipping the weak exit rate comparison for some equivalence classes that contain certain unobservable states:

- An observable state is a state that enables an observable action that cannot be pre-empted by any enabled unobservable action.
- An initially unobservable state is a state in which all the enabled observable actions are pre-empted by some enabled unobservable action, but at least one of the computations starting at this state with one of the higher priority enabled unobservable actions reaches an observable state.
- A fully unobservable state is a state in which all the enabled observable actions are pre-empted by some enabled unobservable action, and all the computations starting at this state with one of the higher priority enabled unobservable actions are unobservable (note that $\underline{0}$ is fully unobservable). We denote by $\mathcal{P}_{\mathrm{CE,fu}}$ the set of the fully unobservable process terms of $\mathcal{P}_{\mathrm{CE}}$.

Definition 32. *An equivalence relation $\mathcal{B} \subseteq \mathcal{P}_{\mathrm{CE}} \times \mathcal{P}_{\mathrm{CE}}$ is a weak extended Markovian bisimulation iff, whenever $(P_1, P_2) \in \mathcal{B}$, then for all action names $a \in Name \cup \{\tau\}$ and priority levels $l \in \mathbb{Z}$ such that no-pre(l, P_1) and no-pre(l, P_2):*

$$rate_{\mathrm{w}}(P_1, a, l, C) = rate_{\mathrm{w}}(P_2, a, l, C) \qquad \text{for all observable } C \in \mathcal{P}_{\mathrm{CE}}/\mathcal{B}$$
$$rate_{\mathrm{w}}(P_1, a, l, \mathcal{P}_{\mathrm{CE,fu}}) = rate_{\mathrm{w}}(P_2, a, l, \mathcal{P}_{\mathrm{CE,fu}})$$ ■

Definition 33. *Weak extended Markovian bisimilarity, denoted by $\approx_{\mathrm{EMB}}$, is the union of all the weak extended Markovian bisimulations.* ■

$\approx_{\mathrm{EMB}}$ enjoys the same properties as $\sim_{\mathrm{EMB}}$ except for congruence. In fact, to recover congruence with respect to parallel composition, we have to restrict ourselves to the set $\mathcal{P}_{\mathrm{CE,wp}}$ of the well-prioritized process terms of $\mathcal{P}_{\mathrm{CE}}$. This is the smallest subset of $\mathcal{P}_{\mathrm{CE}}$ closed under null term, action prefix, alternative composition, and recursion such that the following holds: If $P_1, P_2 \in \mathcal{P}_{\mathrm{CE,wp}}$ and all the

immediate/passive transitions of $[\![P_1]\!]$ (resp. $[\![P_2]\!]$) have priority level/constraint less than the priority level of any unobservable transition of $[\![P_2]\!]$ (resp. $[\![P_1]\!]$), then $P_1 \parallel_S P_2 \in \mathcal{P}_{\mathrm{CE,wp}}$.

The additional characterizing axioms of $\approx_{\mathrm{EMB}}$ over $\mathcal{P}_{\mathrm{CE,wp}}$ are the following:

$$\begin{array}{r} <a,\lambda>.\sum_{i\in I} <\tau,\infty_{l,w_i}>.P_i = \sum_{i\in I} <a,\lambda\cdot w_i/\,\Sigma_{k\in I}\, w_k>.P_i \\ <a,\infty_{l',w'}>.\sum_{i\in I} <\tau,\infty_{l,w_i}>.P_i = \sum_{i\in I} <a,\infty_{l',w'\cdot w_i/\,\Sigma_{k\in I}\, w_k}>.P_i \\ <a,*^{l'}_{w'}>.\sum_{i\in I} <\tau,\infty_{l,w_i}>.P_i = \sum_{i\in I} <a,*^{l'}_{w'\cdot w_i/\,\Sigma_{k\in I}\, w_k}>.P_i \end{array}$$

which witness the capability of $\approx_{\mathrm{EMB}}$ of abstracting from immediate τ-actions in a way that correctly takes into account their weights.

5 Markovian Testing Equivalence

Markovian testing equivalence considers two process terms to be equivalent whenever an external observer, who can interact with them by means of tests, is not able to distinguish between them from the functional or performance viewpoint. In this section we provide the definition of Markovian testing equivalence over $\mathcal{P}_{\mathrm{C,pc}}$ and we recall its properties from [7,9].

5.1 Test Formalization

The only way that the external observer has to infer information about the behavior of the process terms is to interact with them by means of tests and look at their reactions. Was the test passed? If so, with which probability? And how long did it take to pass the test?

In our Markovian framework with asymmetric action synchronization discipline, the most convenient way to represent a test is through another process term composed of passive actions only, which interacts with the terms to be tested by means of a parallel composition operator that enforces synchronization on any action name. In this way, the parallel composition of a performance closed term to be tested and a test will still be performance closed.

From the testing viewpoint, in any of its states a process term to be tested generates the proposal of an action to be executed by means of a race among the exponentially timed actions enabled in that state. Then the test either reacts by participating in the interaction with the process term through a passive action having the same name as the proposed exponentially timed action, or blocks the interaction if it has no passive actions with the proposed name.

Since it is necessary to measure the probability with which process terms pass tests within a finite amount of time, for the test formalization we can restrict ourselves to non-recursive terms (composed of passive actions only). In other words, the expressiveness provided by labeled multitransition systems with a finite dag-like structure will be enough for the tests.

In order to represent the fact that a test is passed or not, each of the terminal nodes of the dag-like semantic model underlying a test must be suitably labeled so as to establish whether it is a success or failure state. At the process calculus level, this amounts to replace $\underline{0}$ with two zeroary operators, which we denote by "s" (for success) and "f" (for failure).

Ambiguous terms like $\mathrm{s} + \mathrm{f}$ will be avoided in the test syntax by replacing the action prefix operator and the binary alternative composition operator with a set of n-ary guarded alternative composition operators, with n ranging over the whole $\mathbf{N}_{>0}$.

Definition 34. *The set $\mathcal{T}$ of the tests is generated by the following syntax:*

$$\begin{array}{l} T ::= \mathrm{f} \\ \quad \mid \ \mathrm{s} \\ \quad \mid \ \displaystyle\sum_{i \in I} <a_i, *_{w_i}>.T_i \end{array}$$

where I is a non-empty finite index set. ■

5.2 Equivalence Definition

Markovian testing equivalence relies on comparing the process term probabilities of performing a successful test-driven computation within a given sequence of average amounts of time. A test-driven computation is a sequence of transitions in the labeled multitransition system underlying the parallel composition of a process term and a test. Due to the restrictions imposed on the test syntax, all the considered test-driven computations will turn out to have a finite length, hence the inductive definitions of Sect. 2.3 apply to them.

Definition 35. *Let $P \in \mathcal{P}_{\mathrm{C,pc}}$ and $T \in \mathcal{T}$. The interaction system of P and T is process term $P \parallel_{Name} T \in \mathcal{P}_{\mathrm{C,pc}}$, where we say that:*

- *A configuration is a state of $[\![P \parallel_{Name} T]\!]$.*
- *A configuration is formed by a process part and a test part.*
- *A configuration is successful (resp. failed) iff its test part is "s" (resp. "f").*
- *A computation is successful (resp. failed) iff so is its last configuration.*
- *A computation that is neither successful nor failed is interrupted.*

We denote by $\mathcal{SC}(P,T)$ the multiset of the successful computations of $\mathcal{C}_{\mathrm{f}}(P \parallel_{Name} T)$. ■

Note that $\mathcal{SC}(P,T) \subseteq \mathcal{I}_{\mathrm{f}}(P \parallel_{Name} T)$, because of the maximality of the successful test-driven computations, and that $\mathcal{SC}(P,T)$ is finite, because of the finitely-branching structure of the considered terms.

Definition 36. *Let $P_1, P_2 \in \mathcal{P}_{\mathrm{C,pc}}$. We say that P_1 is Markovian testing equivalent to P_2, written $P_1 \sim_{\mathrm{MT}} P_2$, iff for all tests $T \in \mathcal{T}$ and sequences $\theta \in (\mathbf{R}_{>0})^*$ of average amounts of time:*

$$prob(\mathcal{SC}_{\leq\theta}(P_1,T)) = prob(\mathcal{SC}_{\leq\theta}(P_2,T))$$

■

Obviously, $\sim_{\rm MT}$ is strictly finer than classical testing equivalence [21] and probabilistic testing equivalence [17,19]. On the other hand, it is strictly coarser than $\sim_{\rm MB}$ as it is less sensitive to branching points. A consequence of this fact is that the derivatives of two Markovian testing equivalent terms are not necessarily related by $\sim_{\rm MT}$. We conclude with a necessary condition.

Proposition 2. *Let $P_1, P_2 \in \mathcal{P}_{\rm C,pc}$ and $T \in \mathcal{T}$. Whenever $P_1 \sim_{\rm MT} P_2$, then for all $c_k \in \mathcal{SC}(P_k, T)$ with $k \in \{1,2\}$ there exists $c_h \in \mathcal{SC}(P_h, T)$ with $h \in \{1,2\} - \{k\}$ such that:*

$$trace(c_k) = trace(c_h)$$
$$time_{\rm a}(c_k) = time_{\rm a}(c_h)$$

and for all $a \in Name$:

$$rate(P_{k,\rm last}, a, 0, \mathcal{P}_{\rm C}) = rate(P_{h,\rm last}, a, 0, \mathcal{P}_{\rm C})$$

with $P_{k,\rm last}$ (resp. $P_{h,\rm last}$) being the process part of the last configuration of c_k (resp. c_h). ■

5.3 Alternative Characterizations

We now present two alternative characterizations of $\sim_{\rm MT}$. The first one is based on the probability distribution of passing a test within a certain sequence of amounts of time. A consequence of this characterization is that considering the (more accurate) probability distributions quantifying the durations of the test-driven computations leads to the same equivalence as considering the (easier to work with) average durations of the test-driven computations. This justifies the use of expected values instead of random variables in the definition of $\sim_{\rm MT}$.

Definition 37. *Let $P_1, P_2 \in \mathcal{P}_{\rm C,pc}$. We say that P_1 is Markovian distribution-testing equivalent to P_2, written $P_1 \sim_{\rm MT,d} P_2$, iff for all tests $T \in \mathcal{T}$ and sequences $\theta \in (\mathbb{R}_{>0})^*$ of amounts of time:*

$$prob_{\rm d}(\mathcal{SC}(P_1, T), \theta) = prob_{\rm d}(\mathcal{SC}(P_2, T), \theta)$$ ■

Theorem 6. *Let $P_1, P_2 \in \mathcal{P}_{\rm C,pc}$. Then:*

$$P_1 \sim_{\rm MT,d} P_2 \iff P_1 \sim_{\rm MT} P_2$$ ■

The second alternative characterization of $\sim_{\rm MT}$ is based on traces that are suitably extended with the sets of the action names permitted at each step by the environment. This means that it is possible to characterize $\sim_{\rm MT}$ in a way that fully abstracts from the tests. A consequence of the proof of this characterization is the identification of a set of canonical tests, i.e. a set of tests that are necessary and sufficient in order to establish whether two process terms are Markovian testing equivalent. Each such test admits a single computation leading to success, whose states can have additional computations each leading to failure in one step.

Definition 38. *An element σ of $(Name \times 2^{Name})^*$ is an extended trace iff either σ is the empty sequence or:*

$$\sigma \equiv (a_1, \mathcal{E}_1) \circ (a_2, \mathcal{E}_2) \circ \ldots \circ (a_n, \mathcal{E}_n)$$

for some $n \in \mathbb{N}_{>0}$ with $a_i \in \mathcal{E}_i$ for each $i = 1, \ldots, n$. We denote by $\mathcal{ET}$ the set of the extended traces. ■

Definition 39. *Let $\sigma \in \mathcal{ET}$. The trace associated with σ is defined by induction on the length of σ through the following Name*-valued function:*

$$trace(\sigma) = \begin{cases} \varepsilon & \text{if } length(\sigma) = 0 \\ a \circ trace(\sigma') & \text{if } \sigma \equiv (a, \mathcal{E}) \circ \sigma' \end{cases}$$

where ε is the empty trace. ■

Definition 40. *Let $P \in \mathcal{P}_{\mathrm{C,pc}}$, $c \in \mathcal{C}_{\mathrm{f}}(P)$, and $\sigma \in \mathcal{ET}$. We say that c is compatible with σ iff:*

$$trace(c) = trace(\sigma)$$

We denote by $\mathcal{CC}(P, \sigma)$ the multiset of the computations of $\mathcal{C}_{\mathrm{f}}(P)$ that are compatible with σ. ■

Note that $\mathcal{CC}(P, \sigma) \subseteq \mathcal{I}_{\mathrm{f}}(P)$ because of the compatibility of the considered computations with the same extended trace σ.

Definition 41. *Let $P \in \mathcal{P}_{\mathrm{C,pc}}$, $\sigma \in \mathcal{ET}$, and $c \in \mathcal{CC}(P, \sigma)$. The probability of executing c with respect to σ is defined by induction on the length of c through the following $\mathbf{R}_{]0,1]}$-valued function:*

$$prob^{\sigma}(c) = \begin{cases} 1 & \text{if } length(c) = 0 \\ \dfrac{\lambda}{\sum\limits_{b \in \mathcal{E}} rate(P, b, 0, \mathcal{P}_{\mathrm{C}})} \cdot prob^{\sigma'}(c') & \text{if } c \equiv P \xrightarrow{a,\lambda} c' \\ & \text{with } \sigma \equiv (a, \mathcal{E}) \circ \sigma' \end{cases}$$

We also define the probability of executing a computation of C with respect to σ as:

$$prob^{\sigma}(C) = \sum_{c \in C} prob^{\sigma}(c)$$

for all $C \subseteq \mathcal{CC}(P, \sigma)$. ■

Definition 42. *Let $P \in \mathcal{P}_{\mathrm{C,pc}}$, $\sigma \in \mathcal{ET}$, and $c \in \mathcal{CC}(P, \sigma)$. The average duration of c with respect to σ is defined by induction on the length of c through the following $(\mathbf{R}_{>0})^*$-valued function:*

$$time_{\mathrm{a}}^{\sigma}(c) = \begin{cases} \varepsilon & \text{if } length(c) = 0 \\ \dfrac{1}{\sum\limits_{b \in \mathcal{E}} rate(P, b, 0, \mathcal{P}_{\mathrm{C}})} \circ time_{\mathrm{a}}^{\sigma'}(c') & \text{if } c \equiv P \xrightarrow{a,\lambda} c' \\ & \text{with } \sigma \equiv (a, \mathcal{E}) \circ \sigma' \end{cases}$$

where ε is the empty average duration. We also define the multiset of the computations of C whose average duration with respect to σ is not greater than θ as:

$$C^{\sigma}_{\leq \theta} = \{\!|\, c \in C \mid length(c) \leq length(\theta) \wedge \forall i = 1, \ldots, length(c).\, time_{\mathrm{a}}^{\sigma}(c)[i] \leq \theta[i] \,|\!\}$$

for all $C \subseteq \mathcal{CC}(P, \sigma)$ and $\theta \in (\mathbf{R}_{>0})^*$. ■

Definition 43. *Let $P_1, P_2 \in \mathcal{P}_{\mathrm{C,pc}}$. We say that P_1 is Markovian extended-trace equivalent to P_2, written $P_1 \sim_{\mathrm{MTr,e}} P_2$, iff for all extended traces $\sigma \in \mathcal{ET}$ and sequences $\theta \in (\mathbb{R}_{>0})^*$ of average amounts of time:*

$$prob^{\sigma}(\mathcal{CC}^{\sigma}_{\leq\theta}(P_1,\sigma)) = prob^{\sigma}(\mathcal{CC}^{\sigma}_{\leq\theta}(P_2,\sigma))$$

■

Theorem 7. *Let $P_1, P_2 \in \mathcal{P}_{\mathrm{C,pc}}$. Then:*

$$P_1 \sim_{\mathrm{MTr,e}} P_2 \iff P_1 \sim_{\mathrm{MT}} P_2$$

■

Definition 44. *The set $\mathcal{T}_{\mathrm{c}}$ of the canonical tests is generated by the following syntax:*

$$\boxed{\begin{array}{l} T ::= \mathrm{s} \\ \quad \mid \; <a, *_1>.T + \sum\limits_{b \in \mathcal{E}-\{a\}} <b, *_1>.\mathrm{f} \end{array}}$$

where the summation is absent whenever $\mathcal{E} - \{a\} = \emptyset$. ■

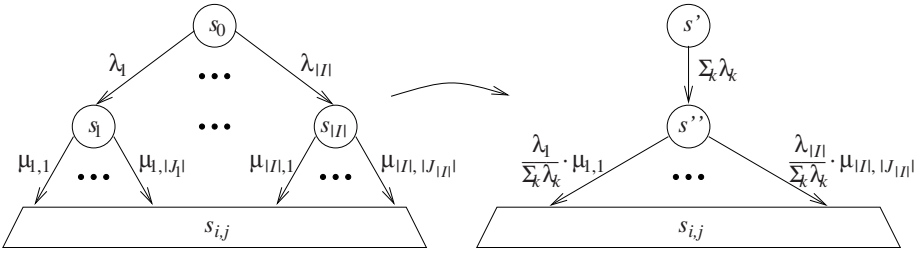
Corollary 2. *Let $P_1, P_2 \in \mathcal{P}_{\mathrm{C,pc}}$. Then $P_1 \sim_{\mathrm{MT}} P_2$ iff for all $T \in \mathcal{T}_{\mathrm{c}}$ and $\theta \in (\mathbb{R}_{>0})^*$:*

$$prob(\mathcal{SC}_{\leq\theta}(P_1,T)) = prob(\mathcal{SC}_{\leq\theta}(P_2,T))$$

■

5.4 Exactness

Markovian testing equivalence induces a CTMC-level aggregation that is strictly coarser than ordinary lumping and was not known in the CTMC field. This aggregation can be depicted through the following rewriting rule:



where:

- I is a finite index set with $|I| \geq 2$.
- k ranges over I.
- J_i is a non-empty finite index set for all $i \in I$.
- For all $i_1, i_2 \in I$:

$$\boxed{\sum_{j \in J_{i_1}} \mu_{i_1,j} = \sum_{j \in J_{i_2}} \mu_{i_2,j}}$$

  with each summation being zero whenever its index set is empty.

Theorem 8. *The CTMC-level aggregation induced by $\sim_{\mathrm{MT}}$ is exact.* ■

5.5 Congruence

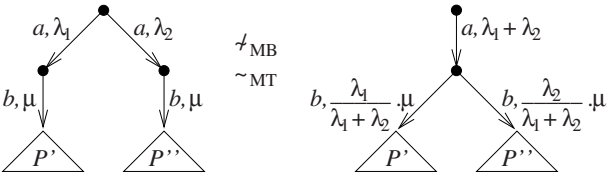
Markovian testing equivalence is a congruence with respect to all the operators of CMPC. In particular, we have what follows.

Theorem 9. *Let $P_1, P_2 \in \mathcal{P}_{\mathrm{C,pc}}$. Whenever $P_1 \sim_{\mathrm{MT}} P_2$, then:*

1. $<a, \lambda>.P_1 \sim_{\mathrm{MT}} <a, \lambda>.P_2$ *for all* $<a, \lambda> \in Act_{\mathrm{S}}$.
2. $P_1 + P \sim_{\mathrm{MT}} P_2 + P$ *and* $P + P_1 \sim_{\mathrm{MT}} P + P_2$ *for all* $P \in \mathcal{P}_{\mathrm{C,pc}}$.
3. $P_1 \parallel_S P \sim_{\mathrm{MT}} P_2 \parallel_S P$ *and* $P \parallel_S P_1 \sim_{\mathrm{MT}} P \parallel_S P_2$ *for all* $S \subseteq Name$ *and* $P \in \mathcal{P}_{\mathrm{C}}$ *containing only passive actions such that* $P_1 \parallel_S P, P_2 \parallel_S P \in \mathcal{P}_{\mathrm{C,pc}}$. ■

5.6 Axiomatization

Markovian testing equivalence is strictly coarser than Markovian bisimilarity, hence the axioms of Table 1 are still valid for $\sim_{\mathrm{MT}}$ over $\mathcal{P}_{\mathrm{C,pc}}$, but not complete. In fact, the two process terms depicted below (with $P' \not\sim_{\mathrm{MB}} P''$):



show that $\sim_{\mathrm{MB}}$ is highly sensitive to branching points. By contrast, $\sim_{\mathrm{MT}}$ allows choices to be deferred as long as they are related to branches starting with actions having the same name that are immediately followed by actions having the same names and the same total rates in all the branches.

The two terms above constitute the simplest instance of an axiom schema subsuming $\mathcal{A}_4^{\mathrm{MB}}$ that characterizes $\sim_{\mathrm{MT}}$. The axiom schema is the following:

$$\sum_{i \in I} <a, \lambda_i>. \sum_{j \in J_i} <b_{i,j}, \mu_{i,j}>.P_{i,j} = <a, \sum_{k \in I} \lambda_k>. \sum_{i \in I} \sum_{j \in J_i} <b_{i,j}, \frac{\lambda_i}{\sum_{k \in I} \lambda_k} \cdot \mu_{i,j}>.P_{i,j}$$

where:

- I is a finite index set with $|I| \geq 2$.
- J_i is a finite index set for all $i \in I$, with the related summation being $\underline{0}$ whenever $J_i = \emptyset$.
- For all $i_1, i_2 \in I$ and $b \in Name$:

$$\sum_{j \in J_{i_1}} \{\!|\, \mu_{i_1,j} \mid b_{i_1,j} = b \,|\!\} = \sum_{j \in J_{i_2}} \{\!|\, \mu_{i_2,j} \mid b_{i_2,j} = b \,|\!\}$$

  with each summation being zero whenever its index set is empty.

5.7 Logical Characterization

Markovian testing equivalence has a modal logic characterization over $\mathcal{P}_{\rm C,pc}$ based on a Markovian variant of a restriction of the Hennessy-Milner logic, in which both negation and logical conjunction are ruled out, while the diamond operator is made dependent from the environment.

Unlike $\rm HML_{MB}$, where the syntax is decorated with rate lower bounds and the satisfaction relation is qualitative, here there is no extra information in the syntax and a quantitative interpretation inspired by [39] is adopted. This establishes the probability with which a process term satisfies a formula quickly enough on average, i.e. within a given sequence of average amounts of time.

Definition 45. *The set of the formulas of* $\rm HML_{MT}$ *is generated by the following syntax:*

$$\boxed{\phi ::= \text{true} \mid \langle a|\mathcal{E}\rangle\phi}$$

where $a \in Name$ *and* $\mathcal{E} \subseteq Name$ *such that* $a \in \mathcal{E}$. ■

Definition 46. *The interpretation function* $[\![.]\!]_{\rm MT}$ *of* $\rm HML_{MT}$ *over* $\mathcal{P}_{\rm C,pc} \times (\mathbb{R}_{>0})^*$ *is defined by structural induction as follows:*

$$\boxed{\begin{array}{l} [\![\text{true}]\!]_{\rm MT}(P,\theta) = 1 \\ [\![\langle a|\mathcal{E}\rangle\phi]\!]_{\rm MT}(P,\theta) = \begin{cases} 0 & \text{if } \theta = \varepsilon \vee \frac{1}{\sum_{b\in\mathcal{E}} rate(P,b,0,\mathcal{P}_{\rm C})} > \theta[1] \\ \sum\limits_{P \xrightarrow{a,\lambda} P'} \frac{\lambda}{\sum_{b\in\mathcal{E}} rate(P,b,0,\mathcal{P}_{\rm C})} \cdot [\![\phi]\!]_{\rm MT}(P',\theta') & \\ \qquad \text{if } \theta = t \circ \theta' \wedge \frac{1}{\sum_{b\in\mathcal{E}} rate(P,b,0,\mathcal{P}_{\rm C})} \leq t \end{cases} \end{array}}$$

where the summation is taken to be zero whenever there are no a*-transitions departing from* P. ■

Theorem 10. *Let* $P_1, P_2 \in \mathcal{P}_{\rm C,pc}$. *Then:*
$P_1 \sim_{\rm MT} P_2 \iff \forall \phi \in \rm HML_{MT}. \forall \theta \in (\mathbb{R}_{>0})^*. [\![\phi]\!]_{\rm MT}(P_1,\theta) = [\![\phi]\!]_{\rm MT}(P_2,\theta)$ ■

5.8 Verification Complexity

Markovian testing equivalence can be decided in polynomial time because two action-labeled CTMCs are Markovian testing equivalent iff their corresponding embedded action-labeled DTMCs (with suitably enriched labels) are probabilistic testing equivalent, with the latter coinciding with probabilistic ready equivalence and hence being decidable in polynomial time [35].

The algorithm to check whether $P_1 \sim_{\rm MT} P_2$ thus proceeds as follows:

1. Transform $[\![P_1]\!]$ and $[\![P_2]\!]$ into their corresponding embedded discrete-time versions:
   (a) Divide the rate of each transition by the total exit rate of its source state.

(b) Augment the name of each transition with the total exit rate of its source state.

2. Compute the equivalence $\mathcal{R}$ that relates any two states of the disjoint union of $[\![P_1]\!]$ and $[\![P_2]\!]$ such that their two sets of (original) action names labeling their outgoing transitions coincide.
3. For each equivalence class R induced by $\mathcal{R}$, apply the algorithm of [47] to check the embedded discrete-time versions of $[\![P_1]\!]$ and $[\![P_2]\!]$ for probabilistic language equivalence by considering R as the set of accepting states.

The time complexity is $O(n^5)$, where n is the number of states of the disjoint union of $[\![P_1]\!]$ and $[\![P_2]\!]$.

6 Markovian Trace Equivalence

Markovian trace equivalence considers two process terms to be equivalent whenever they are able to perform computations with the same functional and performance characteristics. In this section we provide the definition of Markovian trace equivalence over $\mathcal{P}_{\mathrm{C,pc}}$ and we recall its properties from [49,7,9].

6.1 Equivalence Definition

Markovian trace equivalence relies on comparing the process term probabilities of performing a computation within a given sequence of average amounts of time. We emphasize that here, given a process term $P \in \mathcal{P}_{\mathrm{C,pc}}$, we no longer have tests that interact with P. Instead, we directly consider the finite-length computations of P, to which the inductive definitions of Sect. 2.3 apply.

Definition 47. *Let $P \in \mathcal{P}_{\mathrm{C,pc}}$, $c \in \mathcal{C}_{\mathrm{f}}(P)$, and $\alpha \in Name^*$. We say that c is compatible with α iff:*

$$trace(c) = \alpha$$

We denote by $\mathcal{CC}(P,\alpha)$ the multiset of the finite-length computations of P that are compatible with α. ■

Note that $\mathcal{CC}(P,\alpha) \subseteq \mathcal{I}_{\mathrm{f}}(P)$, because of the compatibility of the computations with the same trace α, and that $\mathcal{CC}(P,\alpha)$ is finite, because of the finitely-branching structure of the considered terms.

Definition 48. *Let $P_1, P_2 \in \mathcal{P}_{\mathrm{C,pc}}$. We say that P_1 is Markovian trace equivalent to P_2, written $P_1 \sim_{\mathrm{MTr}} P_2$, iff for all traces $\alpha \in Name^*$ and sequences $\theta \in (\mathbb{R}_{>0})^*$ of average amounts of time:*

$$prob(\mathcal{CC}_{\leq\theta}(P_1,\alpha)) = prob(\mathcal{CC}_{\leq\theta}(P_2,\alpha))$$ ■

Obviously, $\sim_{\mathrm{MTr}}$ is strictly finer than classical trace equivalence [33] and probabilistic trace equivalence [36]. On the other hand, it is strictly coarser than $\sim_{\mathrm{MT}}$ as it completely overrides branching points. Thus, similarly to $\sim_{\mathrm{MT}}$, the derivatives of two Markovian trace equivalent terms are not necessarily related by $\sim_{\mathrm{MTr}}$. We conclude with a necessary condition.

Proposition 3. *Let $P_1, P_2 \in \mathcal{P}_{\mathrm{C,pc}}$ and $\alpha \in Name^*$. Whenever $P_1 \sim_{\mathrm{MTr}} P_2$, then for all $c_k \in \mathcal{CC}(P_k, \alpha)$ with $k \in \{1,2\}$ there exists $c_h \in \mathcal{CC}(P_h, \alpha)$ with $h \in \{1,2\} - \{k\}$ such that:*

$$trace(c_k) = trace(c_h)$$
$$time_{\mathrm{a}}(c_k) = time_{\mathrm{a}}(c_h)$$

and:

$$rate_{\mathrm{t}}(P_{k,\mathrm{last}}, 0) = rate_{\mathrm{t}}(P_{h,\mathrm{last}}, 0)$$

with $P_{k,\mathrm{last}}$ (resp. $P_{h,\mathrm{last}}$) being the last configuration of c_k (resp. c_h). ■

6.2 Alternative Characterizations

Similarly to $\sim_{\mathrm{MT}}$, it turns out that $\sim_{\mathrm{MTr}}$ has an alternative characterization based on the probability distribution of executing a trace within a certain sequence of amounts of time. A consequence of this characterization is that considering the (more accurate) probability distributions quantifying the durations of the computations leads to the same equivalence as considering the (easier to work with) average durations of the computations. This justifies the use of expected values instead of random variables in the definition of $\sim_{\mathrm{MTr}}$.

Definition 49. *Let $P_1, P_2 \in \mathcal{P}_{\mathrm{C,pc}}$. We say that P_1 is Markovian distribution-trace equivalent to P_2, written $P_1 \sim_{\mathrm{MTr,d}} P_2$, iff for all traces $\alpha \in Name^*$ and sequences $\theta \in (\mathbf{R}_{>0})^*$ of amounts of time:*

$$prob_{\mathrm{d}}(\mathcal{CC}(P_1, \alpha), \theta) = prob_{\mathrm{d}}(\mathcal{CC}(P_2, \alpha), \theta)$$ ■

Theorem 11. *Let $P_1, P_2 \in \mathcal{P}_{\mathrm{C,pc}}$. Then:*

$$P_1 \sim_{\mathrm{MTr,d}} P_2 \iff P_1 \sim_{\mathrm{MTr}} P_2$$ ■

6.3 Other Markovian Trace-Based Equivalences

Like for classical trace equivalence, it is possible to define some variants of $\sim_{\mathrm{MTr}}$, which are based on the notions of completed trace, failure set, ready set, failure trace, and ready trace, respectively.

These variants were originally conceived to overcome some drawbacks of classical trace equivalence. Completed traces are traces ending up in deadlock states, so considering them apart is useful to make classical trace equivalence sensitive to deadlock. A failure set is a set of names of actions that cannot be executed in a certain state, and a failure trace is a trace extended at each step with a failure set. Considering failures makes classical trace equivalence sensitive to safety properties. Likewise, a ready set is the set of the names of all the actions that must be executable in a certain state, and a ready trace is a trace extended at each step with a ready set. Considering readies makes classical trace equivalence sensitive to liveness properties.

Definition 50. *Let $P \in \mathcal{P}_{\mathrm{C,pc}}$, $c \in \mathcal{C}_{\mathrm{f}}(P)$, and $\alpha \in Name^*$. We say that c is a maximal computation compatible with α iff $c \in \mathcal{CC}(P, \alpha)$ and the last configuration of c is deadlocked. We denote by $\mathcal{MCC}(P, \alpha)$ the multiset of the finite-length maximal computations of P that are compatible with α.* ■

Definition 51. *Let $P_1, P_2 \in \mathcal{P}_{\mathrm{C,pc}}$. We say that P_1 is Markovian completed-trace equivalent to P_2, written $P_1 \sim_{\mathrm{MTr,c}} P_2$, iff for all traces $\alpha \in Name^*$ and sequences $\theta \in (\mathbb{R}_{>0})^*$ of average amounts of time:*

$$prob(\mathcal{CC}_{\leq\theta}(P_1,\alpha)) = prob(\mathcal{CC}_{\leq\theta}(P_2,\alpha))$$
$$prob(\mathcal{MCC}_{\leq\theta}(P_1,\alpha)) = prob(\mathcal{MCC}_{\leq\theta}(P_2,\alpha))$$

■

Definition 52. *Let $P \in \mathcal{P}_{\mathrm{C,pc}}$, $c \in \mathcal{C}_{\mathrm{f}}(P)$, and $\varphi \equiv (\alpha, \mathcal{F}) \in Name^* \times 2^{Name}$. We say that c is a failure computation compatible with φ iff $c \in \mathcal{CC}(P,\alpha)$ and the last configuration of c cannot execute any action whose name belongs to the failure set $\mathcal{F}$. We denote by $\mathcal{FCC}(P,\varphi)$ the multiset of the finite-length failure computations of P that are compatible with φ.* ■

Definition 53. *Let $P_1, P_2 \in \mathcal{P}_{\mathrm{C,pc}}$. We say that P_1 is Markovian failure equivalent to P_2, written $P_1 \sim_{\mathrm{MF}} P_2$, iff for all traces with final failure set $\varphi \in Name^* \times 2^{Name}$ and sequences $\theta \in (\mathbb{R}_{>0})^*$ of average amounts of time:*

$$prob(\mathcal{FCC}_{\leq\theta}(P_1,\varphi)) = prob(\mathcal{FCC}_{\leq\theta}(P_2,\varphi))$$

■

Definition 54. *Let $P \in \mathcal{P}_{\mathrm{C,pc}}$, $c \in \mathcal{C}_{\mathrm{f}}(P)$, and $\rho \equiv (\alpha, \mathcal{R}) \in Name^* \times 2^{Name}$. We say that c is a ready computation compatible with ρ iff $c \in \mathcal{CC}(P,\alpha)$ and the set of the names of all the actions executable by the last configuration of c coincides with the ready set $\mathcal{R}$. We denote by $\mathcal{RCC}(P,\rho)$ the multiset of the finite-length ready computations of P that are compatible with ρ.* ■

Definition 55. *Let $P_1, P_2 \in \mathcal{P}_{\mathrm{C,pc}}$. We say that P_1 is Markovian ready equivalent to P_2, written $P_1 \sim_{\mathrm{MR}} P_2$, iff for all traces with final ready set $\rho \in Name^* \times 2^{Name}$ and sequences $\theta \in (\mathbb{R}_{>0})^*$ of average amounts of time:*

$$prob(\mathcal{RCC}_{\leq\theta}(P_1,\rho)) = prob(\mathcal{RCC}_{\leq\theta}(P_2,\rho))$$

■

Definition 56. *Let $P \in \mathcal{P}_{\mathrm{C,pc}}$, $c \in \mathcal{C}_{\mathrm{f}}(P)$, and $\phi \in (Name \times 2^{Name})^*$. We say that c is a failure-trace computation compatible with ϕ iff c is compatible with the trace component of ϕ and each configuration of c cannot execute any action whose name belongs to the corresponding failure set in the failure component of ϕ. We denote by $\mathcal{FTCC}(P,\phi)$ the multiset of the finite-length failure-trace computations of P that are compatible with ϕ.* ■

Definition 57. *Let $P_1, P_2 \in \mathcal{P}_{\mathrm{C,pc}}$. We say that P_1 is Markovian failure-trace equivalent to P_2, written $P_1 \sim_{\mathrm{MFTr}} P_2$, iff for all failure traces $\phi \in (Name \times 2^{Name})^*$ and sequences $\theta \in (\mathbb{R}_{>0})^*$ of average amounts of time:*

$$prob(\mathcal{FTCC}_{\leq\theta}(P_1,\phi)) = prob(\mathcal{FTCC}_{\leq\theta}(P_2,\phi))$$

■

Definition 58. *Let $P \in \mathcal{P}_{\mathrm{C,pc}}$, $c \in \mathcal{C}_{\mathrm{f}}(P)$, and $\varrho \in (Name \times 2^{Name})^*$. We say that c is a ready-trace computation compatible with ϱ iff c is compatible with the trace component of ϱ and the sets of the names of all the actions executable by the configurations of c coincide with the corresponding ready sets in the ready component of ϱ. We denote by $\mathcal{RTCC}(P,\varrho)$ the multiset of the finite-length ready-trace computations of P that are compatible with ϱ.* ■

Definition 59. *Let $P_1, P_2 \in \mathcal{P}_{\mathrm{C,pc}}$. We say that P_1 is Markovian ready-trace equivalent to P_2, written $P_1 \sim_{\mathrm{MRTr}} P_2$, iff for all ready traces $\varrho \in (Name \times 2^{Name})^*$ and sequences $\theta \in (\mathbb{R}_{>0})^*$ of average amounts of time:*

$$prob(\mathcal{RTCC}_{\leq\theta}(P_1, \varrho)) = prob(\mathcal{RTCC}_{\leq\theta}(P_2, \varrho))$$

■

Similarly to the probabilistic case [36,35], in the Markovian framework trace equivalence becomes deadlock sensitive, hence $\sim_{\mathrm{MTr}}$ coincides with Markovian completed-trace equivalence. Moreover, Markovian failure equivalence coincides with Markovian ready equivalence – with both coinciding with $\sim_{\mathrm{MT}}$ – and Markovian failure-trace equivalence coincides with Markovian ready-trace equivalence. As a consequence, the Markovian linear-time/branching-time spectrum turns out to be more condensed than the nondeterministic one [25].

Theorem 12. *The Markovian linear-time/branching-time spectrum is:*

$$\sim_{\mathrm{MB}} \subset \sim_{\mathrm{MRTr}} = \sim_{\mathrm{MFTr}} \subset \sim_{\mathrm{MR}} = \sim_{\mathrm{MT}} = \sim_{\mathrm{MTr,e}} = \sim_{\mathrm{MF}} \subset \sim_{\mathrm{MTr,c}} = \sim_{\mathrm{MTr}}$$

■

6.4 Exactness

From the point of view of the induced CTMC-level aggregation, nothing changes when moving from $\sim_{\mathrm{MT}}$ to $\sim_{\mathrm{MTr}}$.

Theorem 13. *$\sim_{\mathrm{MTr}}$ induces the same CTMC-level aggregation as $\sim_{\mathrm{MT}}$.* ■

Corollary 3. *The CTMC-level aggregation induced by $\sim_{\mathrm{MTr}}$ is exact.* ■

6.5 Congruence

Markovian trace equivalence is a congruence with respect to all the operators of SMPC. In particular, we have what follows.

Theorem 14. *Let $P_1, P_2 \in \mathcal{P}_{\mathrm{S}}$. Whenever $P_1 \sim_{\mathrm{MTr}} P_2$, then:*

1. *$<a,\lambda>.P_1 \sim_{\mathrm{MTr}} <a,\lambda>.P_2$ for all $<a,\lambda> \in Act_{\mathrm{S}}$.*
2. *$P_1 + P \sim_{\mathrm{MTr}} P_2 + P$ and $P + P_1 \sim_{\mathrm{MTr}} P + P_2$ for all $P \in \mathcal{P}_{\mathrm{S}}$.* ■

Unfortunately, similarly to the probabilistic case [36], $\sim_{\mathrm{MTr}}$ is not a congruence with respect to parallel composition. Consider for instance the following two Markovian trace equivalent process terms:

$$P_1 \equiv <a,\lambda_1>.<b,\mu>.P' + <a,\lambda_2>.<c,\mu>.P''$$
$$P_2 \equiv <a,\lambda_1+\lambda_2>.(<b,\tfrac{\lambda_1}{\lambda_1+\lambda_2}\cdot\mu>.P' + <c,\tfrac{\lambda_2}{\lambda_1+\lambda_2}\cdot\mu>.P'')$$

where $b \neq c$. If we place each of them in the following context:

$$\_ \|_{\{a,b,c\}} <a,*_1>.<b,*_1>.\underline{0}$$

we obtain two performance-closed process terms – which we call Q_1 and Q_2 – that are no longer Markovian trace equivalent.

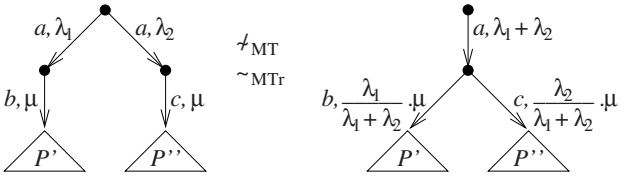
In fact, the following trace:

$$\alpha \equiv a \circ b$$

can distinguish between Q_1 and Q_2. The reason is that the only computation of Q_1 compatible with α is formed by a transition labeled with $<a, \lambda_1>$ followed by a transition labeled with $<b, \mu>$, which has execution probability $\frac{\lambda_1}{\lambda_1+\lambda_2}$ and average duration $\frac{1}{\lambda_1+\lambda_2} \circ \frac{1}{\mu}$. By contrast, the only computation of Q_2 compatible with α is formed by a transition labeled with $<a, \lambda_1 + \lambda_2>$ followed by a transition labeled with $<b, \frac{\lambda_1}{\lambda_1+\lambda_2} \cdot \mu>$, which has execution probability 1 and average duration $\frac{1}{\lambda_1+\lambda_2} \circ \frac{\lambda_1+\lambda_2}{\lambda_1 \cdot \mu}$.

6.6 Axiomatization

Markovian trace equivalence is strictly coarser than Markovian testing equivalence, hence the axioms of $\sim_{\mathrm{MT}}$ are still valid for $\sim_{\mathrm{MTr}}$ over $\mathcal{P}_{\mathrm{S}}$, but not complete. In fact, the two process terms depicted below (with $b \neq c$):



show that, when moving from $\sim_{\mathrm{MT}}$ to $\sim_{\mathrm{MTr}}$, the action prefix operator tends to become left-distributive with respect to the alternative composition operator. More precisely, choices can be deferred as long as they are related to branches starting with actions having the same name that are followed by terms having the same total exit rate. Note that the names and the total rates of the initial actions of such derivative terms can be different in the various branches.

The two terms above constitute the simplest instance of an axiom schema that characterizes $\sim_{\mathrm{MTr}}$, which is more liberal than the one characterizing $\sim_{\mathrm{MT}}$. The axiom schema is the following:

$$\sum_{i \in I} <a, \lambda_i>. \sum_{j \in J_i} <b_{i,j}, \mu_{i,j}>.P_{i,j} = <a, \sum_{k \in I} \lambda_k>. \sum_{i \in I} \sum_{j \in J_i} <b_{i,j}, \frac{\lambda_i}{\Sigma_{k \in I}\, \lambda_k} \cdot \mu_{i,j}>.P_{i,j}$$

where:

- I is a finite index set with $|I| \geq 2$.
- J_i is a finite index set for all $i \in I$, with the related summation being $\underline{0}$ whenever $J_i = \emptyset$.
- For all $i_1, i_2 \in I$:

$$\sum_{j \in J_{i_1}} \mu_{i_1,j} = \sum_{j \in J_{i_2}} \mu_{i_2,j}$$

  with each summation being zero whenever its index set is empty.

6.7 Logical Characterization

Markovian trace equivalence has a modal logic characterization over $\mathcal{P}_{\mathrm{C,pc}}$ similar to the one for Markovian testing equivalence, in which the diamond operator is no longer dependent from the environment.

Definition 60. *The set of the formulas of* $\mathrm{HML}_{\mathrm{MTr}}$ *is generated by the following syntax:*

$$\boxed{\phi ::= \mathrm{true} \mid \langle a \rangle \phi}$$

where $a \in Name$. ■

Definition 61. *The interpretation function* $[\![.]\!]_{\mathrm{MTr}}$ *of* $\mathrm{HML}_{\mathrm{MTr}}$ *over* $\mathcal{P}_{\mathrm{C,pc}} \times (\mathbb{R}_{>0})^*$ *is defined by structural induction as follows:*

$$\boxed{\begin{array}{l} [\![\mathrm{true}]\!]_{\mathrm{MTr}}(P,\theta) = 1 \\ [\![\langle a \rangle \phi]\!]_{\mathrm{MTr}}(P,\theta) = \begin{cases} 0 & \text{if } \theta = \varepsilon \vee \frac{1}{rate_{\mathrm{t}}(P,0)} > \theta[1] \\ \sum\limits_{P \xrightarrow{a,\lambda} P'} \frac{\lambda}{rate_{\mathrm{t}}(P,0)} \cdot [\![\phi]\!]_{\mathrm{MTr}}(P',\theta') & \\ & \text{if } \theta = t \circ \theta' \wedge \frac{1}{rate_{\mathrm{t}}(P,0)} \leq t \end{cases} \end{array}}$$

where the summation is taken to be zero whenever there are no a-transitions departing from P. ■

Theorem 15. *Let* $P_1, P_2 \in \mathcal{P}_{\mathrm{C,pc}}$. *Then:*
$P_1 \sim_{\mathrm{MTr}} P_2 \iff \forall \phi \in \mathrm{HML}_{\mathrm{MTr}}. \forall \theta \in (\mathbb{R}_{>0})^*. [\![\phi]\!]_{\mathrm{MTr}}(P_1,\theta) = [\![\phi]\!]_{\mathrm{MTr}}(P_2,\theta)$ ■

6.8 Verification Complexity

Markovian trace equivalence can be decided in polynomial time because two action-labeled CTMCs are Markovian trace equivalent iff their corresponding embedded action-labeled DTMCs (with suitably enriched labels) are probabilistic trace equivalent, with the latter being decidable in polynomial time [35].

Similarly to the verification of $\sim_{\mathrm{MT}}$, the algorithm to check whether $P_1 \sim_{\mathrm{MTr}} P_2$ thus proceeds as follows:

1. Transform $[\![P_1]\!]$ and $[\![P_2]\!]$ into their corresponding embedded discrete-time versions:
   (a) Divide the rate of each transition by the total exit rate of its source state.
   (b) Augment the name of each transition with the total exit rate of its source state.
2. Apply the algorithm of [47] to check the embedded discrete-time versions of $[\![P_1]\!]$ and $[\![P_2]\!]$ for probabilistic language equivalence by considering each of their states as being an accepting state.

The time complexity is $O(n^4)$, where n is the number of states of the disjoint union of $[\![P_1]\!]$ and $[\![P_2]\!]$.

7 Conclusion

In this survey we have recalled the definitions and the properties of Markovian behavioral equivalences. Besides providing information about the Markovian linear-time/branching-time spectrum, we have compared Markovian bisimilarity, Markovian testing equivalence, and Markovian trace equivalence with respect to a number of criteria, as summarized below:

	exact aggregation	*congruence property*	*sound & complete axiomatization*	*logical characteriz.*	*verification complexity*
$\sim_{\mathrm{MB}}$	*OK*	*OK*	*OK*	*OK*	$O(m \cdot \log n)$
$\sim_{\mathrm{MT}}$	*OK*	*OK*	*OK*	*OK*	$O(n^5)$
$\sim_{\mathrm{MTr}}$	*OK*	OK_{SMPC}	OK_{SMPC}	*OK*	$O(n^4)$

As can be noted, $\sim_{\mathrm{MB}}$ is satisfactory with respect to all the criteria, although it is often too discriminating. A good alternative may be $\sim_{\mathrm{MT}}$, as it encodes the viewpoint of an external observer and is more flexible with respect to branching points. By contrast, $\sim_{\mathrm{MTr}}$ cannot be considered as a valid alternative, as it fails to be a congruence with respect to parallel composition.

It is also worth emphasizing the exactness of the CTMC-level aggregations induced by each of the three considered Markovian behavioral equivalences. This means that $\sim_{\mathrm{MB}}$, $\sim_{\mathrm{MT}}$, and $\sim_{\mathrm{MTr}}$ are all meaningful for performance evaluation purposes. Besides justifying the investigation of the other properties, this can be exploited in practice. For instance, such Markovian behavioral equivalences can be used to aggregate the state space of a model by taking advantage of symmetries within the model [24], or to reduce the state space of a model before applying analysis techniques like model checking [45], without altering the performance properties to be assessed.

We conclude by mentioning some open problems in the field of Markovian behavioral equivalences:

- In our framework based on an asymmetric action synchronization discipline, while $\sim_{\mathrm{MB}}$ is defined over non-performance-closed terms too, this is not the case for $\sim_{\mathrm{MT}}$ and $\sim_{\mathrm{MTr}}$. Finding a way to extend their definitions so that it is still possible to determine the execution probability and the average duration of the computations in the presence of passive transitions is highly desirable.
- The exactness of the CTMC-level aggregations induced by $\sim_{\mathrm{MB}}$, $\sim_{\mathrm{MT}}$, and $\sim_{\mathrm{MTr}}$ establishes a strong connection between concurrency theory and Markov chain theory, which in particular gives rise to a process algebraic characterization of the aggregations themselves. The question here is whether the aggregation induced by $\sim_{\mathrm{MT}}$ and $\sim_{\mathrm{MTr}}$ is the coarsest exact non-trivial one that can be obtained, or whether it can be further extended.
- The set of logical operators necessary to characterize Markovian behavioral equivalences decreases as the discriminating power of the equivalences decreases. However, the logical characterization of $\sim_{\mathrm{MT}}$ relies on a

non-standard variant of the diamond operator. What if we replace the non-standard operator with the standard one and we reintroduce logical conjunction? We claim that this may result in an alternative logical characterization of $\sim_{\mathrm{MT}}$ if the diamonds occurring in a conjunction are independent of each other, i.e. if the names of the related actions are all different [39].

- The verification algorithm for $\sim_{\mathrm{MB}}$ can also be used as a minimization algorithm (with respect to $\sim_{\mathrm{MB}}$), whereas this is not the case for the verification algorithms for $\sim_{\mathrm{MT}}$ and $\sim_{\mathrm{MTr}}$. As far as testing is concerned, it should be investigated whether the algorithm for verifying classical testing equivalence [20] can be adapted to the Markovian framework. The reason is that this algorithm reduces the verification of classical testing equivalence over standard state-transition models to the verification of (a generalization of) classical bisimilarity over transformed models inspired by acceptance trees [28], hence it can be exploited for minimization purposes as well.
- We would like to assess whether $\sim_{\mathrm{MT}}$ can be used also for quantitative analysis. So far, it supports a merely qualitative analysis, in the sense that it only allows one to establish whether two models pass an arbitrary test in the same way. What we envision is the possibility of identifying classes of tests that are related to specific performability measures, which may thus be used to evaluate models with respect to certain indices of interest.
- Finally, it would be interesting to develop weaker versions of $\sim_{\mathrm{MB}}$, $\sim_{\mathrm{MT}}$, and $\sim_{\mathrm{MTr}}$. In this survey we have seen $\approx_{\mathrm{IMB}}$ and $\approx_{\mathrm{EMB}}$, which abstract from invisible immediate actions. However, one could also consider the possibility of abstracting from invisible actions that are exponentially timed, which amounts to understand to what extent exponential delays can be neglected. This issue has been tackled in [32,11,13,4], but none of the proposals seems to induce an exact aggregation at the CTMC level.

References

1. M. Ajmone Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis, *"Modelling with Generalized Stochastic Petri Nets"*, John Wiley & Sons, 1995.
2. J.C.M. Baeten and W.P. Weijland, *"Process Algebra"*, Cambridge University Press, 1990.
3. C. Baier, J.-P. Katoen, and H. Hermanns, *"Approximate Symbolic Model Checking of Continuous Time Markov Chains"*, in Proc. of the *10th Int. Conf. on Concurrency Theory (CONCUR 1999)*, LNCS 1664:146-162, Eindhoven (The Netherlands), 1999.
4. C. Baier, J.-P. Katoen, H. Hermanns, and V. Wolf, *"Comparative Branching-Time Semantics for Markov Chains"*, in Information and Computation 200: 149-214, 2005.
5. S. Balsamo, M. Bernardo, and M. Simeoni, *"Performance Evaluation at the Software Architecture Level"*, in *Formal Methods for Software Architectures*, LNCS 2804:207-258, 2003.

6. J.A. Bergstra, A. Ponse, and S.A. Smolka (eds.), *"Handbook of Process Algebra"*, Elsevier, 2001.
7. M. Bernardo, *"Non-Bisimulation-Based Markovian Behavioral Equivalences"*, to appear in Journal of Logic and Algebraic Programming.
8. M. Bernardo and A. Aldini, *"Weak Markovian Bisimilarity: Abstracting from Prioritized/Weighted Internal Immediate Actions"*, submitted for publication.
9. M. Bernardo and S. Botta, *"A Survey of Modal Logics Characterizing Behavioral Equivalences for Nondeterministic and Stochastic Systems"*, to appear in Mathematical Structures in Computer Science.
10. M. Bernardo and M. Bravetti, *"Performance Measure Sensitive Congruences for Markovian Process Algebras"*, in Theoretical Computer Science 290:117-160, 2003.
11. M. Bernardo and R. Cleaveland, *"A Theory of Testing for Markovian Processes"*, in Proc. of the *11th Int. Conf. on Concurrency Theory (CONCUR 2000)*, LNCS 1877:305-319, State College (PA), 2000.
12. H. Bohnenkamp, P.R. D'Argenio, H. Hermanns, and J.-P. Katoen, *"*MODEST*: A Compositional Modeling Formalism for Hard and Softly Timed Systems"*, in IEEE Trans. on Software Engineering 32:812-830, 2006.
13. M. Bravetti, *"Revisiting Interactive Markov Chains"*, in Proc. of the *3rd Int. Workshop on Models for Time-Critical Systems (MTCS 2002)*, ENTCS 68(5):1-20, Brno (Czech Republic), 2002.
14. M. Bravetti, M. Bernardo, and R. Gorrieri, *"A Note on the Congruence Proof for Recursion in Markovian Bisimulation Equivalence"*, in Proc. of the *6th Int. Workshop on Process Algebra and Performance Modelling (PAPM 1998)*, pp. 153-164, Nice (France), 1998.
15. P. Buchholz, *"Exact and Ordinary Lumpability in Finite Markov Chains"*, in Journal of Applied Probability 31:59-75, 1994.
16. P. Buchholz, *"Markovian Process Algebra: Composition and Equivalence"*, in Proc. of the *2nd Int. Workshop on Process Algebra and Performance Modelling (PAPM 1994)*, Technical Report 27-4, pp. 11-30, Erlangen (Germany), 1994.
17. I. Christoff, *"Testing Equivalences and Fully Abstract Models for Probabilistic Processes"*, in Proc. of the *1st Int. Conf. on Concurrency Theory (CONCUR 1990)*, LNCS 458:126-140, Amsterdam (The Netherlands), 1990.
18. G. Clark, S. Gilmore, and J. Hillston, *"Specifying Performance Measures for PEPA"*, in Proc. of the *5th AMAST Int. Workshop on Formal Methods for Real Time and Probabilistic Systems (ARTS 1999)*, LNCS 1601:211-227, Bamberg (Germany), 1999.
19. R. Cleaveland, Z. Dayar, S.A. Smolka, and S. Yuen, *"Testing Preorders for Probabilistic Processes"*, in Information and Computation 154:93-148, 1999.
20. R. Cleaveland and M. Hennessy, *"Testing Equivalence as a Bisimulation Equivalence"*, in Formal Aspects of Computing 5:1-20, 1993.
21. R. De Nicola and M. Hennessy, *"Testing Equivalences for Processes"*, in Theoretical Computer Science 34:83-133, 1983.
22. R. De Nicola, D. Latella, and M. Massink, *"Formal Modeling and Quantitative Analysis of* KLAIM*-Based Mobile Systems"*, in Proc. of the *20th ACM Symp. on Applied Computing (SAC 2005)*, ACM Press, pp. 428-435, Santa Fe (NM), 2005.
23. S. Derisavi, H. Hermanns, and W.H. Sanders, *"Optimal State-Space Lumping in Markov Chains"*, in Information Processing Letters 87:309-315, 2003.

24. S. Gilmore, J. Hillston, and M. Ribaudo, *"An Efficient Algorithm for Aggregating PEPA Models"*, in IEEE Trans. on Software Engineering 27:449-464, 2001.
25. R.J. van Glabbeek, *"The Linear Time - Branching Time Spectrum I"*, in [6], pp. 3-99, 2001.
26. R.J. van Glabbeek, S.A. Smolka, and B. Steffen, *"Reactive, Generative and Stratified Models of Probabilistic Processes"*, in Information and Computation 121:59-80, 1995.
27. N. Götz, U. Herzog, and M. Rettelbach, *"Multiprocessor and Distributed System Design: The Integration of Functional Specification and Performance Analysis Using Stochastic Process Algebras"*, in Proc. of the *16th Int. Symp. on Computer Performance Modelling, Measurement and Evaluation (PERFORMANCE 1993)*, LNCS 729:121-146, Roma (Italy), 1993.
28. M. Hennessy, *"Acceptance Trees"*, in Journal of the ACM 32:896-928, 1985.
29. M. Hennessy and R. Milner, *"Algebraic Laws for Nondeterminism and Concurrency"*, in Journal of the ACM 32:137-162, 1985.
30. H. Hermanns, *"Interactive Markov Chains"*, LNCS 2428, 2002.
31. H. Hermanns and M. Rettelbach, *"Syntax, Semantics, Equivalences, and Axioms for MTIPP"*, in Proc. of the *2nd Int. Workshop on Process Algebra and Performance Modelling (PAPM 1994)*, Technical Report 27-4, pp. 71-87, Erlangen (Germany), 1994.
32. J. Hillston, *"A Compositional Approach to Performance Modelling"*, Cambridge University Press, 1996.
33. C.A.R. Hoare, *"Communicating Sequential Processes"*, Prentice Hall, 1985.
34. R.A. Howard, *"Dynamic Probabilistic Systems"*, John Wiley & Sons, 1971.
35. D.T. Huynh and L. Tian, *"On Some Equivalence Relations for Probabilistic Processes"*, in Fundamenta Informaticae 17:211-234, 1992.
36. C.-C. Jou and S.A. Smolka, *"Equivalences, Congruences, and Complete Axiomatizations for Probabilistic Processes"*, in Proc. of the *1st Int. Conf. on Concurrency Theory (CONCUR 1990)*, LNCS 458:367-383, Amsterdam (The Netherlands), 1990.
37. P.C. Kanellakis and S.A. Smolka, *"CCS Expressions, Finite State Processes, and Three Problems of Equivalence"*, in Information and Computation 86:43-68, 1990.
38. L. Kleinrock, *"Queueing Systems"*, John Wiley & Sons, 1975.
39. M.Z. Kwiatkowska and G.J. Norman, *"A Testing Equivalence for Reactive Probabilistic Processes"*, in Proc. of the *2nd Int. Workshop on Expressiveness in Concurrency (EXPRESS 1998)*, ENTCS 16(2):114-132, Nice (France), 1998.
40. K.G. Larsen and A. Skou, *"Bisimulation through Probabilistic Testing"*, in Information and Computation 94:1-28, 1991.
41. R. Milner, *"Communication and Concurrency"*, Prentice Hall, 1989.
42. R. Paige and R.E. Tarjan, *"Three Partition Refinement Algorithms"*, in SIAM Journal on Computing 16:973-989, 1987.
43. D. Park, *"Concurrency and Automata on Infinite Sequences"*, in Proc. of the *5th GI Conf. on Theoretical Computer Science*, LNCS 104:167-183, 1981.
44. C. Priami, *"Stochastic π-Calculus"*, in Computer Journal 38:578-589, 1995.
45. J. Sproston and S. Donatelli, *"Backward Bisimulation in Markov Chain Model Checking"*, in IEEE Trans. on Software Engineering 32:531-546, 2006.
46. E.W. Stark, R. Cleaveland, and S.A. Smolka, *"A Process-Algebraic Language for Probabilistic I/O Automata"*, in Proc. of the *14th Int. Conf. on Concurrency Theory (CONCUR 2003)*, LNCS 2761:189-203, Marseille (France), 2003.

47. W.-G. Tzeng, *"A Polynomial-Time Algorithm for the Equivalence of Probabilistic Automata"*, in SIAM Journal on Computing 21:216-227, 1992.
48. M. Woodside, *"From Annotated Software Designs (UML SPT/MARTE) to Model Formalisms"*, to appear in *Formal Methods for Performance Evaluation*, LNCS 4486, 2007.
49. V. Wolf, C. Baier, and M. Majster-Cederbaum, *"Trace Machines for Observing Continuous-Time Markov Chains"*, in Proc. of the *3rd Int. Workshop on Quantitative Aspects of Programming Languages (QAPL 2005)*, ENTCS 153(2):259-277, Edinburgh (UK), 2005.

Stochastic Model Checking*

Marta Kwiatkowska, Gethin Norman, and David Parker

School of Computer Science, University of Birmingham
Edgbaston, Birmingham B15 2TT, United Kingdom

Abstract. This tutorial presents an overview of model checking for both discrete and continuous-time Markov chains (DTMCs and CTMCs). Model checking algorithms are given for verifying DTMCs and CTMCs against specifications written in probabilistic extensions of temporal logic, including quantitative properties with rewards. Example properties include the probability that a fault occurs and the expected number of faults in a given time period. We also describe the practical application of stochastic model checking with the probabilistic model checker PRISM by outlining the main features supported by PRISM and three real-world case studies: a probabilistic security protocol, dynamic power management and a biological pathway.

1 Introduction

Probability is an important component in the design and analysis of software and hardware systems. In distributed algorithms electronic coin tossing is used as a symmetry breaker and as a means to derive efficient algorithms, for example in randomised leader election [38,26], randomised consensus [3,18] and root contention in IEEE 1394 FireWire [37,47]. Traditionally, probability has also been used as a tool to analyse system performance, where typically queueing theory is applied to obtain steady-state probabilities in order to arrive at estimates of measures such as throughput and mean waiting time [30,61]. Probability is also used to model unreliable or unpredictable behaviour, as in e.g. fault-tolerant systems and multi-media protocols, where properties such as frame loss of 1 in every 100 can be described probabilistically.

In this tutorial, we summarise the theory and practice of stochastic model checking. There are a number of probabilistic models, of which we will consider two in detail. The first, discrete-time Markov chains (DTMCs), admit *probabilistic choice*, in the sense that one can specify the probability of making a transition from one state to another. Second, we consider continuous-time Markov chains (CTMCs), frequently used in performance analysis, which model *continuous real time* and probabilistic choice: one can specify the rate of making a transition from one state to another. Probabilistic choice, in this model, arises through *race conditions* when two or more transitions in a state are enabled.

* Partly supported by EPSRC grants EP/D07956X and EP/D076625 and Microsoft Research Cambridge contract MRL 2005-44.

M. Bernardo and J. Hillston (Eds.): SFM 2007, LNCS 4486, pp. 220–270, 2007.

Stochastic model checking is a method for calculating the likelihood of the occurrence of certain events during the execution of a system. Conventional model checkers input a description of a model, represented as a state transition system, and a specification, typically a formula in some temporal logic, and return 'yes' or 'no', indicating whether or not the model satisfies the specification. In common with conventional model checking, stochastic model checking involves reachability analysis of the underlying transition system, but, in addition, it must entail the calculation of the actual likelihoods through appropriate numerical or analytical methods.

The specification language is a *probabilistic* temporal logic, capable of expressing temporal relationships between events and likelihood of events and usually obtained from standard temporal logics by replacing the standard path quantifiers with a probabilistic quantifier. For example, we can express the probability of a fault occurring in a given time period during execution, rather than whether it is possible for such a fault to occur. As a specification language for DTMCs we use the temporal logic called Probabilistic Computation Tree Logic (PTCL) [29], which is based on well-known branching-time Computation Tree Logic (CTL) [20]. In the case of CTMCs, we employ the temporal logic Continuous Stochastic Logic (CSL) developed originally by Aziz et al. [4,5] and since extended by Baier et al. [10], also based on CTL.

Algorithms for stochastic model checking were originally introduced in [62,23,29,5,10], derive from conventional model checking, numerical linear algebra and standard techniques for Markov chains. We describe algorithms for PCTL and CSL and for extensions of these logics to specify reward-based properties, giving suitable examples. This is followed by a description of the PRISM model checker [36,53] which implements these algorithms and the outcome of three case studies that were performed with PRISM.

Outline. We first review a number of preliminary concepts in Section 2. Section 3 introduces DTMCs and PCTL model checking while Section 4 considers CTMCs and CSL model checking. Section 5 gives an overview of the probabilistic model checker PRISM and case studies that use stochastic model checking. Section 6 concludes the tutorial.

2 Preliminaries

In the following, we assume some familiarity with probability and measure theory, see for example [16].

Definition 1. *Let Ω be an arbitrary non-empty set and $\mathcal{F}$ a family of subsets of Ω. We say that $\mathcal{F}$ is a* field *on Ω if:*

1. *the empty set $\varnothing$ is in $\mathcal{F}$;*
2. *whenever A is an element of $\mathcal{F}$, then the complement $\Omega \setminus A$ is in $\mathcal{F}$;*
3. *whenever A and B are elements of $\mathcal{F}$, then $A \cup B$ is in $\mathcal{F}$.*

A field of subsets $\mathcal{F}$ is called a σ-algebra *if it is field which is closed under countable union: whenever $A_i \in \mathcal{F}$ for $i \in \mathbb{N}$, then $\cup_{i \in \mathbb{N}} A_i$ is also in $\mathcal{F}$.*

The elements of a σ-algebra are called *measurable sets*, and $(\Omega, \mathcal{F})$ is called a *measurable space*. A σ-algebra *generated* by a family of sets $\mathcal{A}$, denoted $\sigma(\mathcal{A})$, is the smallest σ-algebra that contains $\mathcal{A}$ which exists by the following proposition.

Proposition 1. *For any non-empty set Ω and $\mathcal{A}$ a family of subsets of Ω, there exists a unique smallest σ-algebra containing $\mathcal{A}$.*

Definition 2. *Let $(\Omega, \mathcal{F})$ be a measurable space. A function $\mu : \mathcal{F} \to [0, 1]$ is a* probability measure *on $(\Omega, \mathcal{F})$ and $(\Omega, \mathcal{F}, \mu)$ a* probability space*, if μ satisfies the following properties:*

1. $\mu(\Omega) = 1$
2. $\mu(\cup_i A_i) = \sum_i \mu(A_i)$ *for any countable disjoint sequence $A_1, A_2, \ldots$ of $\mathcal{F}$.*

The measure μ is also referred to as a *probability distribution*. The set Ω is called the *sample space*, and the elements of $\mathcal{F}$ *events*.

In order to go from a notion of size defined on a family of subsets $\mathcal{A}$ to an actual measure on the σ-algebra generated by $\mathcal{A}$, we need an extension theorem. The following [16] is a typical example.

Definition 3. *A family $\mathcal{F}$ of subsets of Ω is called a* semi-ring *if*

1. *the empty set $\varnothing$ is in $\mathcal{F}$;*
2. *whenever A and B are elements of $\mathcal{F}$, then $A \cap B$ is also in $\mathcal{F}$;*
3. *if $A \subseteq B$ are in $\mathcal{F}$, then there are* finitely *many* pairwise disjoint *subsets $C_1, \ldots, C_k \in \mathcal{F}$ such that $B \setminus A = \cup_{i=1}^{k} C_i$.*

This is not the form of the definition most commonly used in field because of the strange last condition but it is precisely the property that holds for 'hyper-rectangles' in $\mathbb{R}^n$ and, more importantly here, for the *cylinder sets* defined later in this tutorial.

Theorem 1. *If $\mathcal{F}$ is a semi-ring on X and $\mu : \mathcal{F} \to [0, \infty]$ satisfies*

1. $\mu(\varnothing) = 0$
2. $\mu(\cup_{i=1}^{k} A_i) = \sum_{i=1}^{k} \mu(A_i)$ *for any finite disjoint sequence $A_1, \ldots, A_k \in \mathcal{F}$*
3. $\mu(\cup_i A_i) \leqslant \sum_i \mu(A_i)$ *for any countable sequence $A_1, A_2, \ldots \in \mathcal{F}$,*

then μ extends to a unique measure on the σ-algebra generated by $\mathcal{F}$.

The proof of this theorem may be found in a standard text on probability and measure, for example [16]. It is straightforward to check that the 'measures' we define on cylinder sets later in the tutorial satisfy the hypotheses of the above theorem. Hence, these can be extended to measures used for the interpretation of the logics PCTL and CSL without ambiguity.

Definition 4. *Let $(\Omega, \mathcal{F}, \mu)$ be a* probability space*. A function $X : \Omega \to \mathbb{R}_{\geqslant 0}$ is said to be a* random variable*.*

Given a random variable $X : \Omega \to \mathbb{R}$ and the probability space $(\Omega, \mathcal{F}, \mu)$ the expectation or average value with respect to the measure μ is given by the following integral:

$$E[X] \stackrel{\text{def}}{=} \int_{\omega \in \Omega} X(\omega) \, \mathrm{d}\mu \, .$$

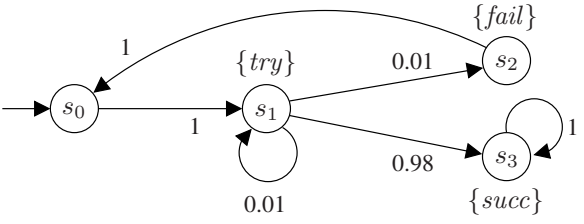


Fig. 1. The four state DTMC $\mathcal{D}_1$

3 Model Checking Discrete-Time Markov Chains

In this section we give an overview of the probabilistic model checking of *discrete-time Markov chains* (DTMCs). Let AP be a fixed, finite set of atomic propositions used to label states with properties of interest.

Definition 5. *A (labelled)* DTMC $\mathcal{D}$ *is a tuple* $(S, \bar{s}, \mathbf{P}, L)$ *where*

- S *is a finite set of* states*;*
- $\bar{s} \in S$ *is the* initial state*;*
- $\mathbf{P} : S \times S \to [0, 1]$ *is the* transition probability matrix *where* $\sum_{s' \in S} \mathbf{P}(s, s') = 1$ *for all* $s \in S$*;*
- $L : S \to 2^{AP}$ *is a* labelling *function which assigns to each state* $s \in S$ *the set* $L(s)$ *of atomic propositions that are valid in the state.*

Each element $\mathbf{P}(s, s')$ of the transition probability matrix gives the probability of making a transition from state s to state s'. Note that the probabilities on transitions emanating from a single state must sum to one. Terminating states, i.e. those from which the system cannot move to another state, can be modelled by adding a self-loop (a single transition going back to the same state with probability 1).

Example 1. Fig. 1 shows a simple example of a DTMC $\mathcal{D}_1 = (S_1, \bar{s}_1, \mathbf{P}_1, L_1)$. In our graphical notation, states are drawn as circles and transitions as arrows, labelled with their associated probabilities. The initial state is indicated by an additional incoming arrow. The DTMC $\mathcal{D}_1$ has four states: $S_1 = \{s_0, s_1, s_2, s_3\}$, with initial state $\bar{s} = s_0$. The transition probability matrix $\mathbf{P}_1$ is given by:

$$\mathbf{P}_1 = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0.01 & 0.01 & 0.98 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

The atomic propositions used to label states are taken from the set $AP = \{try, fail, succ\}$. Here, the DTMC models a simple process which tries to send a message. After one time-step, it enters the state s_1 from which, with probability 0.01 it waits another time-step, with probability 0.98 it successfully sends

the message, and with probability 0.01 it tries but fails to send the message. In the latter case, the process restarts. The labelling function allows us to assign meaningful names to states of the DTMC:

$$L_1(s_0) = \varnothing, \quad L_1(s_1) = \{try\}, \quad L_1(s_2) = \{fail\} \quad \text{and} \quad L_1(s_3) = \{succ\}\,.$$

3.1 Paths and Probability Measures

An execution of a DTMC $\mathcal{D} = (S, \bar{s}, \mathbf{P}, L)$ is represented by a *path*. Formally, a path ω is a non-empty sequence of states $s_0 s_1 s_2 \ldots$ where $s_i \in S$ and $\mathbf{P}(s_i, s_{i+1}) > 0$ for all $i \geqslant 0$. A path can be either finite or infinite. We denote by $\omega(i)$ the ith state of a path ω, $|\omega|$ the length of ω (number of transitions) and for a finite path ω_{fin}, the last state by $last(\omega_{fin})$. We say that a finite path ω_{fin} of length n is a *prefix* of the infinite path ω if $\omega_{fin}(i) = \omega(i)$ for $0 \leqslant i \leqslant n$. The sets of all infinite and finite paths of $\mathcal{D}$ starting in state s are denoted $Path^{\mathcal{D}}(s)$ and $Path^{\mathcal{D}}_{fin}(s)$, respectively. Unless stated explicitly, we always deal with infinite paths.

In order to reason about the probabilistic behaviour of the DTMC, we need to determine the probability that certain paths are taken. This is achieved by defining, for each state $s \in S$, a probability measure Pr_s over the set of infinite paths $Path^{\mathcal{D}}(s)$. Below, we give an outline of this construction. For further details, see [41]. The probability measure is induced by the transition probability matrix $\mathbf{P}$ as follows. For any finite path $\omega_{fin} \in Path^{\mathcal{D}}_{fin}(s)$, we define the probability $\mathbf{P}_s(\omega_{fin})$:

$$\mathbf{P}_s(\omega_{fin}) \stackrel{\text{def}}{=} \begin{cases} 1 & \text{if } n = 0 \\ \mathbf{P}(\omega(0), \omega(1)) \cdots \mathbf{P}(\omega(n-1), \omega(n)) & \text{otherwise} \end{cases}$$

where $n = |\omega_{fin}|$. Next, we define the *cylinder set* $C(\omega_{fin}) \subseteq Path^{\mathcal{D}}(s)$ as:

$$C(\omega_{fin}) \stackrel{\text{def}}{=} \{\omega \in Path^{\mathcal{D}}(s) \,|\, \omega_{fin} \text{ is a prefix of } \omega\}$$

that is, the set of all infinite paths with prefix ω_{fin}. Then, let $\Sigma_{Path^{\mathcal{D}}(s)}$ be the smallest σ-algebra (see Section 2) on $Path^{\mathcal{D}}(s)$ which contains all the sets $C(\omega_{fin})$, where ω_{fin} ranges over the finite paths $Path^{\mathcal{D}}_{fin}(s)$. As the set of cylinders form a semi-ring over $(Path^{\mathcal{D}}(s), \Sigma_{Path^{\mathcal{D}}(s)})$, we can apply Theorem 1 and define Pr_s on $(Path^{\mathcal{D}}(s), \Sigma_{Path^{\mathcal{D}}(s)})$ as the unique measure such that:

$$Pr_s(C(\omega_{fin})) = \mathbf{P}_s(\omega_{fin}) \text{ for all } \omega_{fin} \in Path^{\mathcal{D}}_{fin}(s)\,.$$

Note that, since $C(s) = Path^{\mathcal{D}}(s)$ and $\mathbf{P}_s(s) = 1$, it follows that Pr_s is a probability measure. We can now quantify the probability that, starting from a state $s \in S$, the DTMC $\mathcal{D}$ behaves in a specified fashion by identifying the set of paths which satisfy this specification and, assuming that this set is measurable, using the measure Pr_s.

Example 2. Consider again the DTMC $\mathcal{D}_1$ in Example 1 (see Fig. 1). There are five distinct paths of length 3 starting in state s_0. The probability measure of the cylinder sets associated with each of these is:

$$\begin{aligned}
Pr_{s_0}(C(s_0s_1s_1s_1)) &= 1.00 \cdot 0.01 \cdot 0.01 = 0.0001\\
Pr_{s_0}(C(s_0s_1s_1s_2)) &= 1.00 \cdot 0.01 \cdot 0.01 = 0.0001\\
Pr_{s_0}(C(s_0s_1s_1s_3)) &= 1.00 \cdot 0.01 \cdot 0.98 = 0.0098\\
Pr_{s_0}(C(s_0s_1s_2s_0)) &= 1.00 \cdot 0.01 \cdot 1.00 = 0.01\\
Pr_{s_0}(C(s_0s_1s_3s_3)) &= 1.00 \cdot 0.98 \cdot 1.00 = 0.98\,.
\end{aligned}$$

3.2 Probabilistic Computation Tree Logic (PCTL)

Specifications for DTMC models can be written in PCTL (Probabilistic Computation Tree Logic) [29], a probabilistic extension of the temporal logic CTL. PCTL is essentially the same as the logic pCTL of [6].

Definition 6. *The syntax of PCTL is as follows:*

$$\begin{aligned}
\Phi &::= \mathtt{true} \mid a \mid \neg\Phi \mid \Phi \wedge \Phi \mid \mathtt{P}_{\sim p}[\phi]\\
\phi &::= \mathtt{X}\ \Phi \mid \Phi\ \mathtt{U}^{\leqslant k}\ \Phi
\end{aligned}$$

where a is an atomic proposition, $\sim \in \{<, \leqslant, \geqslant, >\}$, $p \in [0,1]$ and $k \in \mathbb{N} \cup \{\infty\}$.

PCTL formulae are interpreted over the states of a DTMC. For the presentation of the syntax, above, we distinguish between state formulae Φ and path formulae ϕ, which are evaluated over states and paths, respectively. To specify a property of a DTMC, we always use a state formula: path formulae only occur as the parameter of the $\mathtt{P}_{\sim p}[\cdot]$ operator. Intuitively, a state s of $\mathcal{D}$ satisfies $\mathtt{P}_{\sim p}[\phi]$ if the probability of taking a path from s satisfying ϕ is in the interval specified by $\sim p$. For this, we use the probability measure Pr_s over $(Path^{\mathcal{D}}(s), \Sigma_{Path^{\mathcal{D}}(s)})$ introduced in the previous section.

As path formulae we allow the $\mathtt{X}$ ('next') and $\mathtt{U}^{\leqslant k}$ ('bounded until') operators which are standard in temporal logic. The unbounded until is obtained by taking k equal to ∞, i.e. $\Phi\ \mathtt{U}\ \Psi = \Phi\ \mathtt{U}^{\leqslant\infty}\ \Psi$.

Intuitively, $\mathtt{X}\ \Phi$ is true if Φ is satisfied in the next state and $\Phi\ \mathtt{U}^{\leqslant k}\ \Psi$ is true if Ψ is satisfied within k time-steps and Φ is true up until that point.

For a state s and PCTL formula Φ, we write $s \models \Phi$ to indicate that s satisfies Φ. Similarly, for a path ω satisfying path formula ϕ, we write $\omega \models \phi$. The semantics of PCTL over DTMCs is defined as follows.

Definition 7. *Let $\mathcal{D} = (S, \bar{s}, \mathbf{P}, L)$ be a labelled DTMC. For any state $s \in S$, the satisfaction relation $\models$ is defined inductively by:*

$$\begin{aligned}
s \models \mathtt{true} \quad & \textit{for all } s \in S\\
s \models a \quad &\Leftrightarrow \quad a \in L(s)\\
s \models \neg\Phi \quad &\Leftrightarrow \quad s \not\models \Phi\\
s \models \Phi \wedge \Psi \quad &\Leftrightarrow \quad s \models \Phi \wedge s \models \Psi\\
s \models \mathtt{P}_{\sim p}[\phi] \quad &\Leftrightarrow \quad Prob^{\mathcal{D}}(s, \phi) \sim p
\end{aligned}$$

where:

$$Prob^{\mathcal{D}}(s,\phi) \stackrel{\text{def}}{=} Pr_s\{\omega \in Path^{\mathcal{D}}(s) \mid \omega \models \phi\}$$

and for any path $\omega \in Path^{\mathcal{D}}(s)$:

$$\begin{array}{rcl} \omega \models \mathtt{X}\ \Phi & \Leftrightarrow & \omega(1) \models \Phi \\ \omega \models \phi\ \mathtt{U}^{\leqslant k}\ \Psi & \Leftrightarrow & \exists i \in \mathbb{N}.\,(\, i \leqslant k \wedge \omega(i) \models \Psi \wedge \forall j{<}i.\,(\omega(j) \models \Phi)\,)\,. \end{array}$$

Note that, for any state s and path formula ϕ, the set $\{\omega \in Path^{\mathcal{D}}(s) \mid \omega \models \phi\}$ is a *measurable* set of $(Path^{\mathcal{D}}(s), \Sigma_{Path^{\mathcal{D}}(s)})$, see for example [62], and hence Pr_s is well defined over this set. From the basic syntax of PCTL, given above, we can derive a number of additional useful operators. Among these are the well known logical equivalences:

$$\begin{aligned} \mathtt{false} &\equiv \neg \mathtt{true} \\ \Phi \vee \Psi &\equiv \neg(\neg\Phi \wedge \neg\Psi) \\ \Phi \rightarrow \Psi &\equiv \neg\Phi \vee \Psi\,. \end{aligned}$$

We also allow path formulae to contain the $\Diamond$ ('diamond' or 'eventually') operator, which is common in temporal logic. Intuitively, $\Diamond\Phi$ means that Φ is eventually satisfied and its bounded variant $\Diamond^{\leqslant k}\Phi$ means that Φ is satisfied within k time units. These can be expressed in terms of the PCTL 'until' operator as follows:

$$\begin{aligned} \mathtt{P}_{\sim p}[\Diamond\ \Phi] &\equiv \mathtt{P}_{\sim p}[\mathtt{true}\ \mathtt{U}^{\leqslant \infty}\ \Phi] \\ \mathtt{P}_{\sim p}[\Diamond^{\leqslant k}\ \Phi] &\equiv \mathtt{P}_{\sim p}[\mathtt{true}\ \mathtt{U}^{\leqslant k}\ \Phi]\,. \end{aligned}$$

Another common temporal logic operator is $\Box$ ('box' or 'always'). A path satisfies $\Box\Phi$ when Φ is true in every state of the path. Similarly, the bounded variant $\Box^{\leqslant k}\Phi$ means that Φ is true in the first k states of the path. In theory, one can express $\Box$ in terms of $\Diamond$ as follows:

$$\begin{aligned} \Box\Phi &\equiv \neg\Diamond\neg\Phi \\ \Box^{\leqslant k}\Phi &\equiv \neg\Diamond^{\leqslant k}\neg\Phi\,. \end{aligned}$$

However, the syntax of PCTL does not allow negation of path formulae. Observing, though, that for a state s and path formula Φ, $Prob^{\mathcal{D}}(s,\neg\phi) = 1 - Prob^{\mathcal{D}}(s,\phi)$, we can show for example that:

$$\begin{aligned} \mathtt{P}_{\geqslant p}[\Box\ \Phi] &\Leftrightarrow Prob^{\mathcal{D}}(s, \Box\ \Phi) \geqslant p \\ &\Leftrightarrow 1 - Prob^{\mathcal{D}}(s, \Diamond\ \neg\Phi) \geqslant p \\ &\Leftrightarrow Prob^{\mathcal{D}}(s, \Diamond\ \neg\Phi) \leqslant 1 - p \\ &\Leftrightarrow \mathtt{P}_{\leqslant 1-p}[\Diamond\ \neg\Phi]\,. \end{aligned}$$

In fact, we have the following equivalences:

$$\begin{aligned} \mathtt{P}_{\sim p}[\Box\ \Phi] &\equiv \mathtt{P}_{\overline{\sim} 1-p}[\Diamond\ \neg\Phi] \\ \mathtt{P}_{\sim p}[\Box^{\leqslant k}\ \Phi] &\equiv \mathtt{P}_{\overline{\sim} 1-p}[\Diamond^{\leqslant k}\ \neg\Phi] \end{aligned}$$

where $\overline{<} \equiv >$, $\overline{\leqslant} \equiv \geqslant$,$\overline{\geqslant} \equiv \leqslant$ and $\overline{>} \equiv <$.

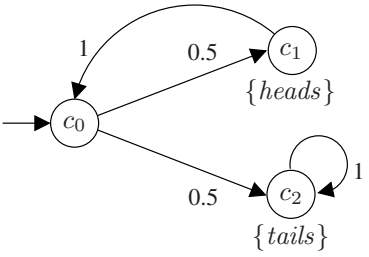


Fig. 2. Example demonstrating difference between $\mathtt{P}_{\geqslant 1}[\Diamond \Phi]$ and $\forall \Diamond \Phi$

The $\mathtt{P}_{\sim p}[\cdot]$ operator of PCTL can be seen as the probabilistic analogue of the path quantifiers of CTL. For example, the PCTL formula $\mathtt{P}_{\sim p}[\Diamond\ \Phi]$, which states that the probability of reaching a Φ-state is $\sim p$, is closely related to the CTL formulae $\forall \Diamond\, \Phi$ and $\exists \Diamond\, \Phi$ (sometimes written AF Φ and EF Φ), which assert that *all* paths or *at least one* path reach a Φ-state, respectively. In fact, we have the following equivalence:

$$\exists \Diamond \Phi \equiv \mathtt{P}_{>0}[\Diamond\ \Phi]$$

as the probability is greater than zero if and only if there exists a finite path leading to a Φ-state. Conversely, $\forall \Diamond\, \Phi$ and $\mathtt{P}_{\geqslant 1}[\Diamond\ \Phi]$ are not equivalent. For example, consider the DTMC in Fig. 2 which models a process which repeatedly tosses a fair coin until the result is 'tails'. State c_0 satisfies $\mathtt{P}_{\geqslant 1}[\Diamond\ \mathit{tails}]$ since the probability of reaching c_2 is one. There is, though, an (infinite) path $c_0c_1c_0c_1\ldots$ which never reaches state c_2. Hence, $\forall \Diamond\ \mathit{tails}$ is not satisfied in state c_0.

Example 3. Below are some typical examples of PCTL formulae:

- $\mathtt{P}_{\geqslant 0.4}[\mathtt{X}\ \mathit{delivered}]$ - the probability that a message has been delivered after one time-step is at least 0.4;
- $\mathit{init} \rightarrow \mathtt{P}_{\leqslant 0}[\Diamond\ \mathit{error}]$ - from any initial configuration, the probability that the system reaches an error state is 0;
- $\mathtt{P}_{\geqslant 0.9}[\neg \mathit{down}\ \mathtt{U}\ \mathit{served}]$ - the probability the system does not go down until after the request has been served is at least 0.9;
- $\mathtt{P}_{<0.1}[\neg \mathit{done}\ \mathtt{U}^{\leqslant 10}\ \mathit{fault}]$ - the probability that a fault occurs before the protocol finishes and within 10 time-steps is strictly less than 0.1.

A perceived weakness of PCTL is that it is not possible to determine the actual probability with which a certain path formula is satisfied, only whether or not the probability meets a particular bound. In fact, this restriction is in place purely to ensure that each PCTL formula evaluates to a Boolean. In practice, this constraint can be relaxed: if the outermost operator of a PCTL formula is $\mathtt{P}_{\sim p}$, we can omit the bound $\sim p$ and simply compute the probability instead. Since the algorithm for PCTL model checking proceeds by computing the actual probability and then comparing it to the bound, no additional work is needed. It is also often useful to study a range of such values by varying one or more parameters, either of the model or of the property. Both these observations can be seen in practice in Section 5.

3.3 Model Checking PCTL

We now summarise a model checking algorithm for PCTL over DTMCs, which was first presented in [22,29,23]. The inputs to the algorithm are a labelled DTMC $\mathcal{D} = (S, \bar{s}, \mathbf{P}, L)$ and a PCTL formula Φ. The output is the set of states $Sat(\Phi) = \{s \in S \mid s \models \Phi\}$, i.e. the set containing all the states of the model which satisfy Φ. In a typical scenario, we may only be interested in whether the initial state $\bar{s}$ of the DTMC satisfies Φ. However, the algorithm works by checking whether each state in S satisfies the formula.

The overall structure of the algorithm is identical to the model checking algorithm for CTL [20], the non-probabilistic temporal logic on which PCTL is based. We first construct the parse tree of the formula Φ. Each node of this tree is labelled with a subformula of Φ, the root node is labelled with Φ itself and leaves of the tree will be labelled with either $\mathtt{true}$ or an atomic proposition a. Working upwards towards the root of the tree, we recursively compute the set of states satisfying each subformula. By the end, we have determined whether each state in the model satisfies Φ. The algorithm for PCTL formulae can be summarised as follows:

$$\begin{aligned} Sat(\mathtt{true}) &= S \\ Sat(a) &= \{s \mid a \in L(s)\} \\ Sat(\neg\Phi) &= S \backslash Sat(\Phi) \\ Sat(\Phi \wedge \Psi) &= Sat(\Phi) \cap Sat(\Psi) \\ Sat(\mathtt{P}_{\sim p}[\phi]) &= \{s \in S \mid Prob^{\mathcal{D}}(s, \phi) \sim p\}\,. \end{aligned}$$

Model checking for the majority of these formulae is trivial to implement and is, in fact, the same as for the non-probabilistic logic CTL. The exceptions are formulae of the form $\mathtt{P}_{\sim p}[\phi]$. For these, we must calculate, for all states s of the DTMC, the probability $Prob^{\mathcal{D}}(s, \phi)$ and then compare these values to the bound in the formula. In the following sections, we describe how to compute these values for the two cases: $\mathtt{P}_{\sim p}[\mathtt{X}\ \Phi]$ and $\mathtt{P}_{\sim p}[\Phi\ \mathtt{U}^{\leqslant k}\ \Psi]$. Because of the recursive nature of the PCTL model checking algorithm, we can assume that the relevant sets, $Sat(\Phi)$ or $Sat(\Phi)$ and $Sat(\Psi)$, are already known.

$\mathtt{P}_{\sim p}[\mathtt{X}\ \Phi]$ formulae. In this case, we need to compute the probability $Prob^{\mathcal{D}}(s, \mathtt{X}\ \Phi)$ for each state $s \in S$. This requires the probabilities of the immediate transitions from s:

$$Prob^{\mathcal{D}}(s, \mathtt{X}\ \Phi) = \sum_{s' \in Sat(\Phi)} \mathbf{P}(s, s')\,.$$

We determine the vector $\underline{Prob}^{\mathcal{D}}(\mathtt{X}\ \Phi)$ of probabilities for all states as follows. Assuming that we have a state-indexed column vector $\underline{\Phi}$ with

$$\underline{\Phi}(s) = \begin{cases} 1 & \text{if } s \in Sat(\Phi) \\ 0 & \text{otherwise,} \end{cases}$$

then $\underline{Prob}^{\mathcal{D}}(\mathtt{X}\ \Phi)$ is computed using the single matrix-vector multiplication:

$$\underline{Prob}^{\mathcal{D}}(\mathtt{X}\ \Phi) = \mathbf{P} \cdot \underline{\Phi}\,.$$

Example 4. Consider the PCTL formula $\mathtt{P}_{\geqslant 0.9}[\mathtt{X}\ (\neg try \vee succ)]$ and the DTMC $\mathcal{D}_1$ from Fig. 1. Proceeding recursively from the innermost subformulae, we compute:

$$\begin{aligned} Sat(try) &= \{s_1\} \\ Sat(succ) &= \{s_3\} \\ Sat(\neg succ) &= S \setminus Sat(succ) = \{s_0, s_1, s_2\} \\ Sat(try \wedge \neg succ) &= Sat(try) \cap Sat(\neg succ) = \{s_1\} \cap \{s_0, s_1, s_2\} = \{s_1\} \\ Sat(\neg try \vee succ) &= Sat(\neg(try \wedge \neg succ)) = S \setminus Sat(try \wedge \neg succ) = \{s_0, s_2, s_3\}\,. \end{aligned}$$

This leaves the $\mathtt{X}$ operator, and from above we have $\underline{Prob}^{\mathcal{D}}(\mathtt{X}\ \neg try \vee succ)$ equals:

$$\mathbf{P}_1 \cdot \underline{\neg try \vee succ} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0.01 & 0.01 & 0.98 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 0 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0.99 \\ 1 \\ 1 \end{pmatrix},$$

and hence $Sat(\mathtt{P}_{\geqslant 0.9}[\mathtt{X}\ (\neg try \vee succ)]) = \{s_1, s_2, s_3\}$.

$\mathtt{P}_{\sim p}[\Phi\ \mathtt{U}^{\leqslant k}\ \Psi]$ formulae. For such formulae we need to determine the probabilities $Prob^{\mathcal{D}}(s, \Phi\ \mathtt{U}^{\leqslant k}\ \Psi)$ for all states s where $k \in \mathbb{N} \cup \{\infty\}$. We begin by considering the case when $k \in \mathbb{N}$.

Case when $k \in \mathbb{N}$. This amounts to computing the solution of the following set of equations. For $s \in S$ and $k \in \mathbb{N}$: $Prob^{\mathcal{D}}(s, \Phi\ \mathtt{U}^{\leqslant k}\ \Psi)$ equals

$$\begin{cases} 1 & \text{if } s \in Sat(\Psi) \\ 0 & \text{if } k{=}\,0 \text{ or } s \in Sat(\neg\Phi \wedge \neg\Psi) \\ \sum_{s' \in S} \mathbf{P}(s, s') \cdot Prob^{\mathcal{D}}(s', \Phi\ \mathtt{U}^{\leqslant k-1}\ \Psi) & \text{otherwise.} \end{cases} \quad (1)$$

We now show how these probabilities can be expressed in terms of the transient probabilities of a DTMC. We denote by $\pi^{\mathcal{D}}_{s,k}(s')$ the transient probability in $\mathcal{D}$ of being in state s' after k steps when starting in s, that is:

$$\pi^{\mathcal{D}}_{s,k}(s') = \Pr_s\{\omega \in Path^{\mathcal{D}}(s) \mid \omega(k) = s'\}\,,$$

and require the following PCTL driven transformation of DTMCs.

Definition 8. *For any DTMC $\mathcal{D} = (S, \bar{s}, \mathbf{P}, L)$ and PCTL formula Φ, let $\mathcal{D}[\Phi] = (S, \bar{s}, \mathbf{P}[\Phi], L)$ where, if $s \not\models \Phi$, then $\mathbf{P}[\Phi](s, s') = \mathbf{P}(s, s')$ for all $s' \in S$, and if $s \models \Phi$, then $\mathbf{P}[\Phi](s, s) = 1$ and $\mathbf{P}[\Phi](s, s') = 0$ for all $s' \neq s$.*

Using the transient probabilities and this transformation we can characterise the probabilities $Prob^{\mathcal{D}}(s, \Phi\ \mathtt{U}^{\leqslant k}\ \Psi)$ as follows.

Proposition 2. *For any DTMC $\mathcal{D} = (S, \bar{s}, \mathbf{P}, L)$, state $s \in S$, PCTL formulae Φ and Ψ, and $k \in \mathbb{N}$:*

$$Prob^{\mathcal{D}}(s, \Phi\ \mathtt{U}^{\leqslant k}\ \Psi) = \sum_{s' \models \Psi} \pi^{\mathcal{D}[\neg\Phi \vee \Psi]}_{s,k}(s')\,.$$

Note that $\mathcal{D}[\neg\Phi \vee \Psi] = \mathcal{D}[\neg(\Phi \wedge \Psi)][\Psi]$, that is all states in $Sat(\neg(\Phi \wedge \Psi))$ and $Sat(\Psi)$ are made absorbing (the only transitions available in these states are self-loops). States in $Sat(\neg(\Phi \wedge \Psi))$ are made absorbing because, for any state s in this set, $Prob^{\mathcal{D}}(s, \Phi\ \mathtt{U}^{\leqslant k}\ \Psi)$ is trivially 0 as neither Φ nor Ψ is satisfied in s, since no path starting in s can possibly satisfy the path formula $\Phi\ \mathtt{U}^{\leqslant k}\ \Psi$. On the other hand, states in $Sat(\Psi)$ are made absorbing because, for any state s in this set, we have $Prob^{\mathcal{D}}(s, \Phi\ \mathtt{U}^{\leqslant k}\ \Psi)$ is trivially 1 since all paths leaving s clearly satisfy $\Phi\ \mathtt{U}^{\leqslant k}\ \Psi$.

Using Proposition 2, the vector of probabilities $\underline{Prob}^{\mathcal{D}}(\Phi\ \mathtt{U}^{\leqslant k}\ \Psi)$ can then be computed using the following matrix and vector multiplications:

$$\underline{Prob}^{\mathcal{D}}(\Phi\ \mathtt{U}^{\leqslant k}\ \Psi) = (\mathbf{P}[\neg\Phi \vee \Psi])^k \cdot \underline{\Psi}\,.$$

This product is typically computed in an iterative fashion:

$$\mathbf{P}[\neg\Phi \vee \Psi] \cdot (\cdots(\mathbf{P}[\neg\Phi \vee \Psi] \cdot \underline{\Psi})\cdots)$$

which requires k matrix-vector multiplications. An alternative is to precompute the matrix power $(\mathbf{P}[\neg\Phi \vee \Psi])^k$ and then perform a single matrix-vector multiplication. In theory, this could be more efficient since the matrix power could be computed by repeatedly squaring $\mathbf{P}[\neg\Phi \vee \Psi]$, requiring approximately $log_2 k$, as opposed to k, multiplications. In practice, however, the matrix $\mathbf{P}[\neg\Phi \vee \Psi]$ is large and sparse, and therefore employing such an approach can dramatically increase the number of non-zero matrix elements, having serious implications for both time and memory usage.

Example 5. Let us return to the DTMC $\mathcal{D}_1$ in Fig. 1 and consider the PCTL formula $\mathtt{P}_{>0.98}[\Diamond^{\leqslant 2} succ]$. This is equivalent to the formula $\mathtt{P}_{>0.98}[\mathtt{true}\ \mathtt{U}^{\leqslant 2}\ succ]$. We have:

$$Sat(\mathtt{true}) = \{s_0, s_1, s_2, s_3\} \ \text{ and } \ Sat(succ) = \{s_3\}\,.$$

The matrix $\mathbf{P}_1[\neg\mathtt{true} \vee succ]$, is identical to $\mathbf{P}_1$, and we have that:

$$\begin{aligned}
&\underline{Prob}^{\mathcal{D}_1}(\Phi\ \mathtt{U}^{\leqslant 0}\ \Psi) = \underline{succ} = [0,0,0,1]\\
&\underline{Prob}^{\mathcal{D}_1}(\Phi\ \mathtt{U}^{\leqslant 1}\ \Psi) = \mathbf{P}_1[\neg\mathtt{true} \vee succ] \cdot \underline{Prob}^{\mathcal{D}_1}(\Phi\ \mathtt{U}^{\leqslant 0}\ \Psi) = [0,0.98,0,1]\\
&\underline{Prob}^{\mathcal{D}_1}(\Phi\ \mathtt{U}^{\leqslant 2}\ \Psi) = \mathbf{P}_1[\neg\mathtt{true} \vee succ] \cdot \underline{Prob}^{\mathcal{D}_1}(\Phi\ \mathtt{U}^{\leqslant 1}\ \Psi) = [0.98,0.9898,0,1]\,.
\end{aligned}$$

Hence, $Sat(\mathtt{P}_{>0.98}[\Diamond^{\leqslant 2} succ]) = \{s_1, s_3\}$.

Case when $k = \infty$. Note that, instead of $\mathtt{U}^{\leqslant\infty}$, we use the standard notation $\mathtt{U}$ for unbounded until. The probabilities $\underline{Prob}^{\mathcal{D}}(s, \Phi\ \mathtt{U}\ \Psi)$ can be computed as the least solution of the linear equation system:

$$Prob^{\mathcal{D}}(s, \Phi\ \mathtt{U}\ \Psi) = \begin{cases} 1 & \text{if } s \in Sat(\Psi)\\ 0 & \text{if } s \in Sat(\neg\Phi \wedge \neg\Psi)\\ \sum_{s' \in S} \mathbf{P}(s,s') \cdot Prob^{\mathcal{D}}(s', \Phi\ \mathtt{U}\ \Psi) & \text{otherwise.}\end{cases}$$

PROB0($Sat(\Phi), Sat(\Psi)$)

1. $R := Sat(\Psi)$
2. $done :=$ **false**
3. **while** ($done =$ **false**)
4. $\quad R' := R \cup \{s \in Sat(\Phi) \mid \exists s' \in R \,.\, \mathbf{P}(s,s')>0\}$
5. $\quad$ **if** $(R' = R)$ **then** $done :=$ **true**
6. $\quad R := R'$
7. **endwhile**
8. **return** $S \backslash R$

PROB1($Sat(\Phi), Sat(\Psi), Sat(\mathtt{P}_{\leqslant 0}[\Phi \mathtt{U} \Psi])$)

1. $R := Sat(\mathtt{P}_{\leqslant 0}[\Phi \mathtt{U} \Psi])$
2. $done :=$ **false**
3. **while** ($done =$ **false**)
4. $\quad R' := R \cup \{s \in (Sat(\Phi) \backslash Sat(\Psi)) \mid \exists s' \in R \,.\, \mathbf{P}(s,s')>0\}$
5. $\quad$ **if** $(R' = R)$ **then** $done :=$ **true**
6. $\quad R := R'$
7. **endwhile**
8. **return** $S \backslash R$

Fig. 3. The PROB0 and PROB1 algorithm

To simplify the computation we transform this system of equations into one with a unique solution. This is achieved by first finding *all* the states s for which $Prob^{\mathcal{D}}(s, \Phi \mathtt{U} \Psi)$ is exactly 0 or 1; more precisely, we compute the sets of states:

$$Sat(\mathtt{P}_{\leqslant 0}[\Phi \mathtt{U} \Psi]) = \{s \in S \mid Prob^{\mathcal{D}}(s, \Phi \mathtt{U} \Psi)=0\}$$
$$Sat(\mathtt{P}_{\geqslant 1}[\Phi \mathtt{U} \Psi]) = \{s \in S \mid Prob^{\mathcal{D}}(s, \Phi \mathtt{U} \Psi)=1\}\,.$$

These sets can be determined with the algorithms PROB0 and PROB1 which are described in Fig. 3:

$$Sat(\mathtt{P}_{\leqslant 0}[\Phi \mathtt{U} \Psi]) = \text{PROB0}(Sat(\Phi), Sat(\Psi))$$
$$Sat(\mathtt{P}_{\geqslant 1}[\Phi \mathtt{U} \Psi]) = \text{PROB1}(Sat(\Phi), Sat(\Psi), Sat(\mathtt{P}_{\leqslant 0}[\Phi \mathtt{U} \Psi]))\,.$$

PROB0 computes all the states from which it is possible, with *non-zero* probability, to reach a state satisfying Ψ without leaving states satisfying Φ. It then subtracts these from S to determine the states which have a *zero* probability. PROB1 first determines the set of states for which the probability is *less than* 1 of reaching a state satisfying Ψ without leaving states satisfying Φ. These are the states from which there is a non-zero probability of reaching a state in $Sat(\mathtt{P}_{\leqslant 0}[\Phi \mathtt{U} \Psi])$, passing only through states satisfying Φ but not Ψ. It then subtracts this set from S to produce $Sat(\mathtt{P}_{\geqslant 1}[\Phi \mathtt{U} \Psi])$. Note that both algorithms are based on the computation of a fixpoint operator, and hence require at most $|S|$ iterations.

The probabilities $Prob^{\mathcal{D}}(s, \Phi \mathtt{U} \Psi)$ can then be computed as the unique solution of the following linear equation system:

$$Prob^{\mathcal{D}}(s, \Phi \,\mathtt{U}\, \Psi) = \begin{cases} 1 & \text{if } s \in Sat(\mathtt{P}_{\geqslant 1}[\Phi \,\mathtt{U}\, \Psi]) \\ 0 & \text{if } s \in Sat(\mathtt{P}_{\leqslant 0}[\Phi \,\mathtt{U}\, \Psi]) \\ \sum_{s' \in S} \mathbf{P}(s, s') \cdot Prob^{\mathcal{D}}(s', \Phi \,\mathtt{U}\, \Psi) & \text{otherwise.} \end{cases}$$

Since the probabilities for the sets of states $Sat(\mathtt{P}_{\geqslant 1}[\Phi \,\mathtt{U}\, \Psi])$ and $Sat(\mathtt{P}_{\leqslant 0}[\Phi \,\mathtt{U}\, \Psi])$ are known, i.e. 1 and 0 respectively, it is possible to solve the linear equation system over only the set of states $S^? = S \setminus (Sat(\mathtt{P}_{\geqslant 1}[\Phi \,\mathtt{U}\, \Psi]) \cup Sat(\mathtt{P}_{\leqslant 0}[\Phi \,\mathtt{U}\, \Psi]))$:

$$Prob^{\mathcal{D}}(s, \Phi \,\mathtt{U}\, \Psi) = \sum_{s' \in S^?} \mathbf{P}(s, s') \cdot Prob^{\mathcal{D}}(s, \Phi \,\mathtt{U}\, \Psi) + \sum_{s' \in Sat(\mathtt{P}_{\geqslant 1}[\Phi \,\mathtt{U}\, \Psi])} \mathbf{P}(s, s')$$

reducing the number of unknowns from $|S|$ to $|S^?|$.

In either case, the linear equation system can be solved by any standard approach. These include direct methods, such as Gaussian elimination and L/U decomposition, or iterative methods, such as Jacobi and Gauss-Seidel. The algorithms Prob0 and Prob1 form the first part of the calculation of $Prob^{\mathcal{D}}(s, \Phi \,\mathtt{U}\, \Psi)$. For this reason, we refer to them as *precomputation algorithms*. For *qualitative* PCTL properties (i.e. where the bound p in the formula $\mathtt{P}_{\sim p}[\Phi \,\mathtt{U}\, \Psi]$ is either 0 or 1) or for cases where $Prob^{\mathcal{D}}(s, \Phi \,\mathtt{U}\, \Psi)$ happens to be either 0 or 1 for all states (i.e. $Sat(\mathtt{P}_{\leqslant 0}[\Phi \,\mathtt{U}\, \Psi]) \cup Sat(\mathtt{P}_{\geqslant 1}[\Phi \,\mathtt{U}\, \Psi]) = S$), it suffices to use these precomputation algorithms alone. For *quantitative* properties with an arbitrary bound p, numerical computation is also usually required. The precomputation algorithms are still valuable in this case. Firstly, they can reduce the number of states for which numerical computation is required. Secondly, they determine the exact probability for the states in $Sat(\mathtt{P}_{\leqslant 0}[\Phi \,\mathtt{U}\, \Psi])$ and $Sat(\mathtt{P}_{\geqslant 1}[\Phi \,\mathtt{U}\, \Psi])$, whereas numerical computation typically computes an approximation and is subject to round-off errors.

Finally we note that, if desired, the Prob1 algorithm can be omitted and $Sat(\mathtt{P}_{\geqslant 1}[\Phi \,\mathtt{U}\, \Psi])$ replaced by $Sat(\Psi)$. The set $Sat(\mathtt{P}_{\leqslant 0}[\Phi \,\mathtt{U}\, \Psi])$, however, must be computed to ensure that the linear equation system has a unique solution.

Example 6. Consider again the DTMC $\mathcal{D}_1$ in Fig. 1 and the PCTL formula $\mathtt{P}_{>0.99}[try \,\mathtt{U}\, succ]$. We have $Sat(try) = \{s_1\}$ and $Sat(succ) = \{s_3\}$. Prob0 determines in two iterations that $Sat(\mathtt{P}_{\leqslant 0}[try \,\mathtt{U}\, succ]) = \{s_0, s_2\}$. Prob1 determines that $Sat(\mathtt{P}_{\geqslant 1}[try \,\mathtt{U}\, succ]) = \{s_3\}$. The resulting linear equation system is:

$$\begin{aligned} Prob^{\mathcal{D}_1}(s_0, try \,\mathtt{U}\, succ) &= 0 \\ Prob^{\mathcal{D}_1}(s_1, try \,\mathtt{U}\, succ) &= 0.01 \cdot Prob^{\mathcal{D}_1}(s_1, try \,\mathtt{U}\, succ) + \\ &\quad\; 0.01 \cdot Prob^{\mathcal{D}_1}(s_2, try \,\mathtt{U}\, succ) + \\ &\quad\; 0.98 \cdot Prob^{\mathcal{D}_1}(s_3, try \,\mathtt{U}\, succ) \\ Prob^{\mathcal{D}_1}(s_2, try \,\mathtt{U}\, succ) &= 0 \\ Prob^{\mathcal{D}_1}(s_3, try \,\mathtt{U}\, succ) &= 1\,. \end{aligned}$$

This yields the solution $\underline{Prob}^{\mathcal{D}}(try \,\mathtt{U}\, succ) = (0, \frac{98}{99}, 0, 1)$ and we see that the formula $\mathtt{P}_{>0.99}[try \,\mathtt{U}\, succ]$ is satisfied only in state s_3.

3.4 Extending DTMCs and PCTL with Rewards

In this section we enhance DTMCs with reward (or cost) structures and extend PCTL to allow for specifications over reward structures. For a DTMC $\mathcal{D} = (S, \bar{s}, \mathbf{P}, L)$ a reward structure $(\underline{\rho}, \boldsymbol{\iota})$ allows one to specify two distinct types of rewards: *state* rewards, which are assigned to states by means of the reward function $\underline{\rho} : S \to \mathbb{R}_{\geqslant 0}$, and *transition* rewards, which are assigned to transitions by means of the reward function $\boldsymbol{\iota} : S \times S \to \mathbb{R}_{\geqslant 0}$. The state reward $\underline{\rho}(s)$ is the reward acquired in state s per time-step, i.e. a reward of $\underline{\rho}(s)$ is incurred if the DTMC is in state s for 1 time-step and the transition reward $\boldsymbol{\iota}(s, s')$ is acquired each time a transition between states s and s' occurs.

A reward structure can be used to represent additional information about the system the DTMC represents, for example the power consumption, number of packets sent or the number of lost requests. Note that state rewards are sometimes called cumulative rewards while transition rewards are sometimes referred to as instantaneous or impulse rewards.

Example 7. Returning to Example 1 which describes the DTMC $\mathcal{D}_1$ of Fig. 1, consider the reward structure $(\underline{\rho}^{\mathcal{D}_1}, \mathbf{0})$, where $\underline{\rho}^{\mathcal{D}_1}(s) = 1$ if $s = s_1$ and equals 0 otherwise. This particular reward structure would be useful when we are interested in the number of time-steps spent in state s_1 or the chance that one is in state s_1 after a certain number of time-steps.

The logic PCTL is extended to allow for the reward properties by means of the following state formulae:

$$\mathtt{R}_{\sim r}[\mathtt{C}^{\leqslant k}] \;\big|\; \mathtt{R}_{\sim r}[\mathtt{I}^{=k}] \;\big|\; \mathtt{R}_{\sim r}[\mathtt{F}\ \Phi]$$

where $\sim \in \{<, \leqslant, \geqslant, >\}$, $r \in \mathbb{R}_{\geqslant 0}$, $k \in \mathbb{N}$ and Φ is a PCTL state formula.

Intuitively, a state s satisfies $\mathtt{R}_{\sim r}[\mathtt{C}^{\leqslant k}]$ if, from state s, the expected reward *cumulated* after k time-steps satisfies $\sim r$; $\mathtt{R}_{\sim r}[\mathtt{I}^{=k}]$ is true if from state s the expected state reward at time-step k meets the bound $\sim r$; and $\mathtt{R}_{\sim r}[\mathtt{F}\ \Phi]$ is true if from state s the expected reward cumulated before a state satisfying Φ is reached meets the bound $\sim r$.

Formally, given a DTMC $\mathcal{D} = (S, \bar{s}, \mathbf{P}, L)$, the semantics of these formulae is defined as follows. For any $s \in S$, $k \in \mathbb{N}$, $r \in \mathbb{R}_{\geqslant 0}$ and PCTL formula Φ:

$$\begin{aligned} s \models \mathtt{R}_{\sim r}[\mathtt{C}^{\leqslant k}] \quad &\Leftrightarrow \quad Exp^{\mathcal{D}}(s, X_{\mathtt{C}^{\leqslant k}}) \sim r \\ s \models \mathtt{R}_{\sim r}[\mathtt{I}^{=k}] \quad &\Leftrightarrow \quad Exp^{\mathcal{D}}(s, X_{\mathtt{I}^{=k}}) \sim r \\ s \models \mathtt{R}_{\sim r}[\mathtt{F}\ \Phi] \quad &\Leftrightarrow \quad Exp^{\mathcal{D}}(s, X_{\mathtt{F}\Phi}) \sim r \end{aligned}$$

where $Exp^{\mathcal{D}}(s, X)$ denotes the expectation of the random variable $X : Path^{\mathcal{D}}(s) \to \mathbb{R}_{\geqslant 0}$ with respect to the probability measure Pr_s, and for any path $\omega = s_0 s_1 s_2 \cdots \in Path^{\mathcal{D}}(s)$:

$$X_{\mathtt{C}^{\leqslant k}}(\omega) \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } k = 0 \\ \sum_{i=0}^{k-1} \underline{\rho}(s_i) + \boldsymbol{\iota}(s_i, s_{i+1}) & \text{otherwise} \end{cases}$$

$$X_{\mathtt{I}^{=k}}(\omega) \stackrel{\text{def}}{=} \underline{\rho}(s_k)$$

$$X_{\mathtt{F}\Phi}(\omega) \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } s_0 \models \Phi \\ \infty & \text{if } \forall i \in \mathbb{N}.\ s_i \not\models \Phi \\ \sum_{i=0}^{\min\{j \mid s_j \models \Phi\}-1} \underline{\rho}(s_i) + \boldsymbol{\iota}(s_i, s_{i+1}) & \text{otherwise.} \end{cases}$$

Example 8. Below are some typical examples of reward based specifications using these formulae:

- $\mathtt{R}_{\leqslant 5.5}[\mathtt{C}^{\leqslant 100}]$ - the expected power consumption within the first 100 time-steps of operation is less than or equal to 5.5;
- $\mathtt{R}_{\geqslant 4}[\mathtt{I}^{=10}]$ - the expected number of messages still to be delivered after 10 time-steps have passed is at least 4;
- $\mathtt{R}_{\geqslant 14}[\mathtt{F}\ \mathit{done}]$ - the expected number of correctly delivered messages is at least 14.

We now consider the computation of the expected values for each of the random variables introduced above.

The random variable $X_{\mathtt{C}^{\leqslant k}}$. In this case the computation of the expected values $\mathit{Exp}^{\mathcal{D}}(s, X_{\mathtt{C}^{\leqslant k}})$ for all $s \in S$ is based on the following set of equations:

$$\mathit{Exp}^{\mathcal{D}}(s, X_{\mathtt{C}^{\leqslant k}}) = \begin{cases} 0 & \text{if } k = 0 \\ \underline{\rho}(s) + \sum_{s' \in S} \mathbf{P}(s, s') \cdot \big(\boldsymbol{\iota}(s, s') + \mathit{Exp}^{\mathcal{D}}(s', X_{\mathtt{C}^{\leqslant k-1}})\big) & \text{otherwise.} \end{cases}$$

More precisely, one can iteratively compute the vector of expected values by means of the following matrix-vector operations:

$$\underline{\mathit{Exp}}^{\mathcal{D}}(X_{\mathtt{C}^{\leqslant k}}) = \begin{cases} \underline{0} & \text{if } k = 0 \\ \underline{\rho} + \mathbf{P} \cdot \big(\boldsymbol{\iota} \cdot \underline{1} + \underline{\mathit{Exp}}^{\mathcal{D}}(X_{\mathtt{C}^{\leqslant k-1}})\big) & \text{otherwise} \end{cases}$$

where $\underline{1}$ denotes a vector with all entries equal to 1.

Example 9. Let us return to the DTMC $\mathcal{D}_1$ of Example 1 (see Fig. 1) and reward structure of Example 9. The PCTL formula $\mathtt{R}_{>1}[\mathtt{C}^{\leqslant k}]$ in this case states that, after k time steps, the expected number of time steps spent in state s_1 is greater than 1. Now from above:

$$\begin{aligned} \underline{\mathit{Exp}}^{\mathcal{D}}(X_{\mathtt{C}^{\leqslant 0}}) &= [0, 0, 0, 0] \\ \underline{\mathit{Exp}}^{\mathcal{D}}(X_{\mathtt{C}^{\leqslant 1}}) &= \underline{\rho} + \mathbf{P} \cdot \big(\boldsymbol{\iota} + \underline{\mathit{Exp}}^{\mathcal{D}}(X_{\mathtt{C}^{\leqslant 0}})\big) \\ &= [0, 1, 0, 0] + \mathbf{P} \cdot \big(\mathbf{0} \cdot \underline{1} + [0, 0, 0, 0]\big) \\ &= [0, 1, 0, 0] \\ \underline{\mathit{Exp}}^{\mathcal{D}}(X_{\mathtt{C}^{\leqslant 2}}) &= \underline{\rho} + \mathbf{P} \cdot \big(\boldsymbol{\iota} + \underline{\mathit{Exp}}^{\mathcal{D}}(X_{\mathtt{C}^{\leqslant 1}})\big) \\ &= [0, 1, 0, 0] + \mathbf{P} \cdot \big(\mathbf{0} \cdot \underline{1} + [0, 1, 0, 0]\big) \\ &= [1, 1.01, 0, 0] \end{aligned}$$

and hence $\mathit{Sat}(\mathtt{R}_{>1}[\mathtt{C}^{\leqslant 2}]) = \{s_1\}$.

The random variable $X_{\mathtt{I}=k}$. In this case the expected value can be computed iteratively through the following set of equations:

$$Exp^{\mathcal{D}}(s, X_{\mathtt{I}=k}) = \begin{cases} \underline{\rho}(s) & \text{if } k = 0 \\ \sum_{s' \in S} \mathbf{P} \cdot Exp^{\mathcal{D}}(s, X_{\mathtt{I}=k-1}) & \text{otherwise.} \end{cases}$$

Therefore, the vector $\underline{Exp}^{\mathcal{D}}(X_{\mathtt{I}=k})$ can be computed by means of the following matrix-vector operations:

$$\underline{Exp}^{\mathcal{D}}(X_{\mathtt{I}=k}) = \begin{cases} \underline{\rho} & \text{if } k = 0 \\ \mathbf{P} \cdot \underline{Exp}^{\mathcal{D}}(X_{\mathtt{I}=k-1}) & \text{otherwise.} \end{cases}$$

Example 10. Returning again to the DTMC $\mathcal{D}_1$ of Example 1 and reward structure of Example 9. In this case, the PCTL formula $\mathtt{R}_{>0}[\mathtt{I}^{=k}]$ specifies that, at time-step k, the expectation of being in state s_1 is greater than 0. We have:

$$\begin{aligned} &\underline{Exp}^{\mathcal{D}}(X_{\mathtt{I}=0}) = [0, 1, 0, 0] \\ &\underline{Exp}^{\mathcal{D}}(X_{\mathtt{I}=1}) = \mathbf{P} \cdot \underline{Exp}^{\mathcal{D}}(X_{\mathtt{I}=0}) = \mathbf{P} \cdot [0, 1, 0, 0] = [1, 0.01, 0, 0] \\ &\underline{Exp}^{\mathcal{D}}(X_{\mathtt{I}=2}) = \mathbf{P} \cdot \underline{Exp}^{\mathcal{D}}(X_{\mathtt{I}=1}) = \mathbf{P} \cdot [1, 0.01, 0, 0] = [0.01, 0.0001, 1, 0]\,. \end{aligned}$$

Hence, the states s_0, s_1 and s_2 satisfy the formula $\mathtt{R}_{>0}[\mathtt{I}^{=2}]$.

The random variable $X_{\mathtt{F}\Phi}$. The expectations in this case are a solution of the following system of linear equations:

$$Exp^{\mathcal{D}}(s, X_{\mathtt{F}\Phi}) = \begin{cases} 0 & \text{if } s \in Sat(\Phi) \\ \underline{\rho}(s) + \sum_{s' \in S} \mathbf{P}(s, s') \cdot \big(\boldsymbol{\iota}(s, s') + Exp^{\mathcal{D}}(s', X_{\mathtt{F}\Phi}) \big) & \text{otherwise.} \end{cases}$$

As above, to simplify the computation this system of equations is transformed into one for which the expectations $Exp^{\mathcal{D}}(s, X_{\mathtt{F}\Phi})$ are the unique solution. To achieve this, we identify the sets of states for which $Exp^{\mathcal{D}}(s, X_{\mathtt{F}\Phi})$ equals ∞. This set of states are simply the set of states for which the probability of reaching a Φ state is less than 1, that is, the set $Sat(\mathtt{P}_{<1}[\Diamond\, \Phi])$. We compute this set using the precomputation algorithms Prob1 and Prob0 described in the previous section and the equivalence $\mathtt{P}_{<1}[\Diamond\, \Phi] \equiv \neg \mathtt{P}_{\geqslant 1}[\Diamond\, \Phi]$. One can then compute $Exp^{\mathcal{D}}(s, X_{\mathtt{F}\Phi})$ as the unique solution of the following linear equation system:

$$Exp^{\mathcal{D}}(s, X_{\mathtt{F}\Phi}) = \begin{cases} 0 & \text{if } s \in Sat(\Phi) \\ \infty & \text{if } s \in Sat(\mathtt{P}_{<1}[\Diamond\, \Phi]) \\ \underline{\rho}(s) + \sum_{s' \in S} \mathbf{P}(s, s') \cdot \big(\boldsymbol{\iota}(s, s') + Exp^{\mathcal{D}}(s', X_{\mathtt{F}\Phi}) \big) & \text{otherwise.} \end{cases}$$

As for 'until' formulae, this can be solved using any standard direct or iterative method.

Example 11. Let us return to $\mathcal{D}_1$ of Example 1 and the reward structure in Example 9. The PCTL formula, $\mathtt{R}_{<1}[\mathtt{F}\ succ]$, in this case, asserts that the expected number of times state s_1 is entered before reaching a state satisfying *succ* is less than 1. Following the procedure outlined above, we compute:

$$\begin{aligned}
Sat(succ) &= \{s_3\} \\
Sat(\mathtt{P}_{<1}[\Diamond\ succ]) &= Sat(\neg \mathtt{P}_{\geqslant 1}[\Diamond\ succ]) \\
&= S \setminus Sat(\mathtt{P}_{\geqslant 1}[\Diamond\ succ]) \\
&= S \setminus \textsc{Prob1}(S, Sat(succ), Sat(\mathtt{P}_{\leqslant 0}[\Diamond\ succ])) \\
&= S \setminus \textsc{Prob1}(S, Sat(succ), \textsc{Prob0}(Sat(\mathtt{true}), Sat(succ))) \\
&= S \setminus \textsc{Prob1}(S, Sat(succ), \varnothing) \\
&= S \setminus \{s_0, s_1, s_2, s_3\} = \varnothing.
\end{aligned}$$

This leads to the linear equation system:

$$\begin{aligned}
Exp^{\mathcal{D}}(s_0, X_{\mathtt{F}succ}) &= 0 + 1.00 \cdot \big(0 + Exp^{\mathcal{D}}(s_1, X_{\mathtt{F}succ})\big) \\
Exp^{\mathcal{D}}(s_1, X_{\mathtt{F}succ}) &= 1 + 0.01 \cdot \big(0 + Exp^{\mathcal{D}}(s_1, X_{\mathtt{F}succ})\big) + 0.01 \cdot \big(0 + Exp^{\mathcal{D}}(s_2, X_{\mathtt{F}succ})\big) \\
Exp^{\mathcal{D}}(s_2, X_{\mathtt{F}succ}) &= 0 + 1.00 \cdot \big(0 + Exp^{\mathcal{D}}(s_0, X_{\mathtt{F}succ})\big) \\
Exp^{\mathcal{D}}(s_3, X_{\mathtt{F}succ}) &= 0
\end{aligned}$$

which has the solution $\overline{Exp^{\mathcal{D}}(X_{\mathtt{F}succ})} = \big(\frac{100}{98}, \frac{100}{98}, \frac{100}{98}, 0\big)$, and hence it follows that $Sat(\mathtt{R}_{<1}[\mathtt{F}\ succ]) = \overline{\{s_3\}}$.

3.5 Complexity of PCTL Model Checking

The overall time complexity for model checking a PCTL formula Φ against a DTMC $\mathcal{D} = (S, \bar{s}, \mathbf{P}, L)$ is linear in $|\Phi|$ and polynomial in $|S|$. The size $|\Phi|$ of a formula Φ is, as defined in [29], equal to the number of logical connectives and temporal operators in the formula plus the sum of the sizes of the temporal operators. Because of the recursive nature of the algorithm, we perform model checking for each of the $|\Phi|$ operators and each one is at worst polynomial in $|S|$. The most expensive of these are the operators $\mathtt{P}_{\sim p}[\Phi\ \mathtt{U}\ \Psi]$ and $\mathtt{R}_{\sim r}[\mathtt{F}\ \Phi]$, for which a system of linear equations of size at most $|S|$ must be solved. This can be done with Gaussian elimination, the complexity of which is cubic in the size of the system. Strictly speaking, the complexity of model checking is also linear in $k_{\max}$, the maximum value of k found in formulae of type $\mathtt{P}_{\sim p}[\Phi\ \mathtt{U}^{\leqslant k}\ \Psi]$, $\mathtt{R}_{\sim r}[\mathtt{C}^{\leqslant k}]$ or $\mathtt{R}_{\sim r}[\mathtt{I}^{=k}]$. In practice, k is usually much smaller than $|S|$.

4 Model Checking Continuous-Time Markov Chains

This section concerns model checking continuous-time Markov chains (CTMCs) against the logic Continuous Stochastic Logic (CSL). While each transition between states in a DTMC corresponds to a discrete time-step, in a CTMC transitions occur in real time. As for the case of DTMCs, we suppose that we have a fixed set of atomic propositions AP.

Definition 9. *A (labelled)* CTMC *is a tuple* $\mathcal{C} = (S, \bar{s}, \mathbf{R}, L)$ *where:*

- S *is a finite set of* states*;*
- $\bar{s} \in S$ *is the* initial state*;*
- $\mathbf{R} : S \times S \to \mathbb{R}_{\geqslant 0}$ *is the* transition rate matrix*;*
- $L : S \to 2^{AP}$ *is a* labelling *function which assigns to each state* $s \in S$ *the set* $L(s)$ *of atomic propositions that are valid in the state.*

The transition rate matrix $\mathbf{R}$ assigns rates to each pair of states in the CTMC, which are used as parameters of the exponential distribution. A transition can only occur between states s and s' if $\mathbf{R}(s, s') > 0$ and, in this case, the probability of this transition being triggered within t time-units equals $1 - e^{-\mathbf{R}(s,s') \cdot t}$. Typically, in a state s, there is more than one state s' for which $\mathbf{R}(s, s') > 0$. This is known as a *race condition*. The first transition to be triggered determines the next state of the CTMC. The time spent in state s, before any such transition occurs, is exponentially distributed with rate $E(s)$, where:

$$E(s) \stackrel{\text{def}}{=} \sum_{s' \in S} \mathbf{R}(s, s') \,.$$

$E(s)$ is known as the *exit rate* of state s. A state s is called *absorbing* if $E(s) = 0$, i.e. if it has no outgoing transitions. We can also determine the actual probability of each state s' being the next state to which a transition is made from state s, independent of the time at which this occurs. This is defined by the following DTMC.

Definition 10. *The* embedded DTMC *of a CTMC* $\mathcal{C} = (S, \bar{s}, \mathbf{R}, L)$ *is the DTMC* $emb(\mathcal{C}) = (S, \bar{s}, \mathbf{P}^{emb(\mathcal{C})}, L)$ *where for* $s, s' \in S$*:*

$$\mathbf{P}^{emb(\mathcal{C})}(s, s') = \begin{cases} \frac{\mathbf{R}(s,s')}{E(s)} & \text{if } E(s) \neq 0 \\ 1 & \text{if } E(s) = 0 \text{ and } s = s' \\ 0 & \text{otherwise.} \end{cases}$$

Using the above definitions, we can consider the behaviour of the CTMC in an alternative way. It will remain in a state s for a delay which is exponentially distributed with rate $E(s)$ and then make a transition. The probability that this transition is to state s' is given by $\mathbf{P}^{emb(\mathcal{C})}(s, s')$.

We also define the following matrix, which will be used when we perform analysis of the CTMC.

Definition 11. *The* infinitesimal generator matrix *for the CTMC* $\mathcal{C} = (S, \bar{s}, \mathbf{R}, L)$ *is the matrix* $\mathbf{Q} : S \times S \to \mathbb{R}$ *defined as:*

$$\mathbf{Q}(s, s') = \begin{cases} \mathbf{R}(s, s') & \text{if } s \neq s' \\ -\sum_{s'' \neq s} \mathbf{R}(s, s'') & \text{otherwise.} \end{cases}$$

Example 12. Fig. 4 shows a simple example of a CTMC $\mathcal{C}_1 = (S_1, \bar{s}_1, \mathbf{R}_1, L)$. The graphical notation is as for DTMCs, except that transitions are now labelled

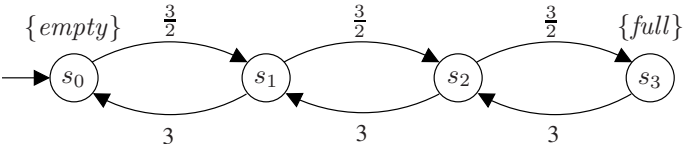


Fig. 4. The four state CTMC $\mathcal{C}_1$

with rates rather than probabilities. The CTMC models a queue of jobs: there are four states s_0, s_1, s_2 and s_3, where state s_i indicates that there are i jobs in the queue. Initially, the queue is empty ($\bar{s} = s_0$) and the maximum size is 3. Jobs arrive with rate $\frac{3}{2}$ and are removed from the queue with rate 3. The associated transition rate matrix $\mathbf{R}_1$, transition probability matrix $\mathbf{P}_1^{emb(\mathcal{C}_1)}$ for the embedded DTMC and infinitesimal generator matrix $\mathbf{Q}_1$ are as follows:

$$\mathbf{R}_1 = \begin{pmatrix} 0 & \frac{3}{2} & 0 & 0 \\ 3 & 0 & \frac{3}{2} & 0 \\ 0 & 3 & 0 & \frac{3}{2} \\ 0 & 0 & 3 & 0 \end{pmatrix} \quad \mathbf{P}_1^{emb(\mathcal{C}_1)} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ \frac{2}{3} & 0 & \frac{1}{3} & 0 \\ 0 & \frac{2}{3} & 0 & \frac{1}{3} \\ 0 & 0 & 1 & 0 \end{pmatrix} \quad \mathbf{Q}_1 = \begin{pmatrix} -\frac{3}{2} & \frac{3}{2} & 0 & 0 \\ 3 & -\frac{9}{2} & \frac{3}{2} & 0 \\ 0 & 3 & -\frac{9}{2} & \frac{3}{2} \\ 0 & 0 & 3 & -3 \end{pmatrix}.$$

We have labelled the state s_0, where the queue contains no jobs, with the atomic proposition *empty* and the state s_3, where it has the maximum number of jobs, with *full*. These are also illustrated in Fig. 4.

4.1 Paths and Probability Measures

An infinite path of a CTMC $\mathcal{C} = (S, \bar{s}, \mathbf{R}, L)$ is a non-empty sequence $s_0 t_0 s_1 s_2 \ldots$ where $\mathbf{R}(s_i, s_{i+1}) > 0$ and $t_i \in \mathbb{R}_{>0}$ for all $i \geqslant 0$. A finite path is a sequence $s_0 t_0 s_1 t_1 s_{k-1} t_{k-1} s_k$ such that s_k is absorbing. The value t_i represents the amount of time spent in the state s_i. As with DTMCs, we denote by $\omega(i)$ the ith state of a path ω, namely s_i. For an infinite path ω, we denote by $time(\omega, i)$ the amount of time spent in state s_i, namely t_i, and by $\omega@t$ the state occupied at time t, i.e. $\omega(j)$ where j is the smallest index for which $\sum_{i=0}^{j} t_i \geqslant t$. For a finite path $\omega = s_0 t_0 s_1 s_2 \ldots t_{k-1} s_k$, $time(\omega, i)$ is only defined for $i \leqslant k$: $time(\omega, j) = t_i$ for $i < k$ and $time(\omega, k) = \infty$. Furthermore, if $t \leqslant \sum_{i=1}^{k-1} t_i$, then $\omega@t$ is defined as for infinite paths, and otherwise $\omega@t = s_k$.

We denote by $Path^{\mathcal{C}}(s)$ the set of all (infinite and finite) paths of the CTMC $\mathcal{C}$ starting in state s. The probability measure Pr_s over $Path^{\mathcal{C}}(s)$, taken from [10], can be defined as follows. If the states $s_0, \ldots, s_n \in S$ satisfy $\mathbf{R}(s_i, s_{i+1}) > 0$ for all $0 \leqslant i < n$ and $I_0, \ldots, I_{n-1}$ are non-empty intervals in $\mathbb{R}_{\geqslant 0}$, then the *cylinder set* $C(s_0, I_0, \ldots, I_{n-1}, s_n)$ is defined to be the set containing all paths $\omega \in Path^{\mathcal{C}}(s_0)$ such that $\omega(i) = s_i$ for all $i \leqslant n$ and $time(\omega, i) \in I_i$ for all $i < n$.

We then let $\Sigma_{Path^{\mathcal{C}}(s)}$ be the smallest σ-algebra on $Path^{\mathcal{C}}(s)$ which contains all the cylinder sets $C(s_0, I_0, \ldots, I_{n-1}, s_n)$, where $s_0, \ldots, s_n \in S$ range over all sequences of states with $s_0 = s$ and $\mathbf{R}(s_i, s_{i+1}) > 0$ for $0 \leqslant i < n$, and $I_0, \ldots, I_{n-1}$ range over all sequences of non-empty intervals in $\mathbb{R}_{\geqslant 0}$. Using Theorem 1, we

can then define the probability measure Pr_s on $\Sigma_{Path^{\mathcal{C}}(s)}$ as the unique measure such that $Pr_s(C(s)) = 1$ and for any cylinder $C(s, I, \ldots, I_{n-1}, s_n, I', s')$, $Pr_s(C(s, I, \ldots, I_{n-1}, s_n, I', s'))$ equals:

$$Pr_s(C(s, I, \ldots, I_{n-1}, s_n)) \cdot \mathbf{P}^{emb(\mathcal{C})}(s_n, s') \cdot \left(e^{-E(s_n)\cdot \inf I'} - e^{-E(s_n)\cdot \sup I'}\right) .$$

Example 13. Consider the CTMC $\mathcal{C}_1$ from Fig. 4 and the sequence of states and intervals $s_0, [0, 2], s_1$ (i.e. taking $I_0 = [0, 2]$ in the notation of the previous paragraph). Using the probability measure Pr_{s_0} over $(Path^{\mathcal{C}_1}(s_0), \Sigma_{Path^{\mathcal{C}}(s_0)})$, for the cylinder set $C(s_0, [0, 2], s_1)$, we have:

$$\begin{aligned} Pr_{s_0}(C(s_0, [0, 2], s_1)) &= Pr_{s_0}(C(s_0)) \cdot \mathbf{P}_1^{emb(\mathcal{C}_1)}(s_0, s_1) \cdot (e^{-E(s_0)\cdot 0} - e^{-E(s_0)\cdot 2}) \\ &= 1 \cdot 1 \cdot (e^0 - e^{-3}) \\ &= 1 - e^{-3} . \end{aligned}$$

Intuitively, this means that the probability of leaving the initial state s_0 and passing to state s_1 within the first 2 time units is $1 - e^{-3} \approx 0.950213$.

4.2 Steady-State and Transient Behaviour

In addition to path probabilities, we consider two more traditional properties of CTMCs: *transient* behaviour, which relates to the state of the model at a particular time instant; and *steady-state* behaviour, which describes the state of the CTMC in the long-run. For a CTMC $\mathcal{C} = (S, \bar{s}, \mathbf{R}, L)$, the transient probability $\pi^{\mathcal{C}}_{s,t}(s')$ is defined as the probability, having started in state s, of being in state s' at time instant t. Using the definitions of the previous section:

$$\pi^{\mathcal{C}}_{s,t}(s') \stackrel{\text{def}}{=} Pr_s\{\omega \in Path^{\mathcal{C}}(s) \,|\, \omega@t = s'\} .$$

The steady-state probability $\pi^{\mathcal{C}}_s(s')$, i.e. the probability of, having started in state s, being in state s' in the long-run, is defined as:

$$\pi^{\mathcal{C}}_s(s') \stackrel{\text{def}}{=} \lim_{t\to\infty} \pi^{\mathcal{C}}_{s,t}(s') .$$

The steady-state probability distribution, i.e. the values $\pi^{\mathcal{C}}_s(s')$ for all $s' \in S$, can be used to infer the percentage of time, in the long-run, that the CTMC spends in each state. For the class of CTMCs which we consider here, i.e. those that are homogeneous and finite-state, the limit in the above definition always exists [59]. Furthermore, for CTMCs which are *irreducible* (strongly connected), that is, those for which there exists a finite path from each of its states to every other state, the steady-state probabilities $\pi^{\mathcal{C}}_s(s')$ are independent of the starting state s.

We now outline a standard technique, called uniformisation (also known as 'randomisation' or 'Jensen's method'), for computing transient probabilities of CTMCs as this will later be relied on in the model checking algorithms for CTMCs.

Uniformisation. For a CTMC $\mathcal{C} = (S, \bar{s}, \mathbf{R}, L)$, we denote by $\boldsymbol{\Pi}_t^{\mathcal{C}}$ the matrix of all transient probabilities for time t, i.e. $\boldsymbol{\Pi}_t^{\mathcal{C}}(s, s') = \pi_{s,t}^{\mathcal{C}}(s')$. It can be shown (see for example [59]) that $\boldsymbol{\Pi}_t^{\mathcal{C}}$ can be expressed as a matrix exponential, and hence evaluated as a power series:

$$\boldsymbol{\Pi}_t^{\mathcal{C}} \;=\; e^{\mathbf{Q}\cdot t} \;=\; \sum_{i=0}^{\infty} \frac{(\mathbf{Q}\cdot t)^i}{i!}$$

where $\mathbf{Q}$ is the infinitesimal generator matrix of $\mathcal{C}$ (see Definition 11). Unfortunately, this computation tends to be unstable. As an alternative, the probabilities can be computed through the *uniformised* DTMC of $\mathcal{C}$.

Definition 12. *For any CTMC $\mathcal{C} = (S, \bar{s}, \mathbf{R}, L)$ with infinitesimal generator matrix $\mathbf{Q}$, the* uniformised *DTMC is given by $unif(\mathcal{C}) = (S, \bar{s}, \mathbf{P}^{unif(\mathcal{C})}, L)$ where $\mathbf{P}^{unif(\mathcal{C})} = \mathbf{I} + \mathbf{Q}/q$ and $q \geqslant \max\{E(s) \mid s \in S\}$.*

The *uniformisation rate* q is determined by the state with the shortest mean residence time. All (exponential) delays in the CTMC $\mathcal{C}$ are normalised with respect to q. That is, for each state $s \in S$ with $E(s) = q$, one epoch in $unif(\mathcal{C})$ corresponds to a single exponentially distributed delay with rate q, after which one of its successor states is selected probabilistically. As a result, no self-loop is added to such states in the DTMC $unif(\mathcal{C})$. If $E(s)<q$ – this state has on average a longer state residence time than $\frac{1}{q}$ – one epoch in $unif(\mathcal{C})$ might not be "long enough". Hence, in the next epoch these states might be revisited and, accordingly, are equipped with a self-loop with probability $1 - \frac{E(s)}{q}$. Note the difference between the embedded DTMC $emb(\mathcal{C})$ and the uniformised DTMC $unif(\mathcal{C})$: whereas the epochs in $\mathcal{C}$ and $emb(\mathcal{C})$ coincide and $emb(\mathcal{C})$ can be considered as the time less variant of $\mathcal{C}$, a single epoch in $unif(\mathcal{C})$ corresponds to a single exponentially distributed delay with rate q in $\mathcal{C}$. Now, using the uniformised DTMC the matrix of transient probabilities can be expressed as:

$$\boldsymbol{\Pi}_t^{\mathcal{C}} \;=\; \sum_{i=0}^{\infty} \gamma_{i,q\cdot t} \cdot \left(\mathbf{P}^{unif(\mathcal{C})}\right)^i \quad \text{where} \quad \gamma_{i,q\cdot t} = e^{-q\cdot t} \cdot \frac{(q\cdot t)^i}{i!}\,. \tag{2}$$

In fact, this reformulation has a fairly intuitive explanation. Each step of the uniformised DTMC corresponds to one exponentially distributed delay, with parameter q, in the CTMC. The matrix power $\left(\mathbf{P}^{unif(\mathcal{C})}\right)^i$ gives the probability of jumping between each pair of states in the DTMC in i steps and $\gamma_{i,q\cdot t}$ is the ith Poisson probability with parameter $q\cdot t$, the probability of i such steps occurring in time t, given the delay is exponentially distributed with rate q.

This approach has a number of important advantages. Firstly, unlike $\mathbf{Q}$, the matrix $\mathbf{P}^{unif(\mathcal{C})}$ is *stochastic*, meaning that all entries are in the range $[0, 1]$ and all rows sum to one. Computations using $\mathbf{P}^{unif(\mathcal{C})}$ are therefore more numerically stable. In particular, $\mathbf{Q}$ contains both positive and negative values which can cause severe round-off errors.

Secondly, the infinite sum is now easier to truncate. For example, the techniques of Fox and Glynn [27], which allow efficient computation of the Poisson

probabilities $\gamma_{i,q\cdot t}$, also produce an upper and lower bound $(L_\varepsilon, R_\varepsilon)$, for some desired precision ε, below and above which the probabilities are insignificant. Hence, the sum can be computed only over this range.

Lastly, the computation can be carried out efficiently using matrix-vector multiplications, rather than more costly matrix-matrix multiplications. Consider the problem of computing $\pi^{\mathcal{C}}_{s,t}(s')$ for a fixed state s. These values can be obtained by pre-multiplying the matrix $\boldsymbol{\Pi}^{\mathcal{C}}_t$ by the initial probability distribution, in this case the vector $\underline{\pi}^{\mathcal{C}}_{s,0}$ where $\pi^{\mathcal{C}}_{s,0}(s')$ is equal to 1 if $s' = s$ and 0 otherwise:

$$\underline{\pi}^{\mathcal{C}}_{s,t} = \underline{\pi}^{\mathcal{C}}_{s,0} \cdot \boldsymbol{\Pi}^{\mathcal{C}}_t = \underline{\pi}^{\mathcal{C}}_{s,0} \cdot \sum_{i=0}^{\infty} \gamma_{i,q\cdot t} \cdot \left(\mathbf{P}^{unif(\mathcal{C})}\right)^i .$$

Rearranging, this can be expressed as a sum of vectors, rather than a sum of matrix powers:

$$\underline{\pi}^{\mathcal{C}}_{s,t} = \sum_{i=0}^{\infty} \left(\gamma_{i,q\cdot t} \cdot \underline{\pi}^{\mathcal{C}}_{s,0} \cdot \left(\mathbf{P}^{unif(\mathcal{C})}\right)^i \right)$$

where the vector required in each element of the summation is computed by a single matrix-vector multiplication, using the vector from the previous iteration:

$$\underline{\pi}^{\mathcal{C}}_{s,0} \cdot \left(\mathbf{P}^{unif(\mathcal{C})}\right)^i = \left(\underline{\pi}^{\mathcal{C}}_{s,0} \cdot \left(\mathbf{P}^{unif(\mathcal{C})}\right)^{i-1} \right) \cdot \mathbf{P}^{unif(\mathcal{C})} .$$

Hence, the total work required is R_ε matrix-vector multiplications.

4.3 Continuous Stochastic Logic (CSL)

We write specifications of CTMCs using the logic CSL (Continuous Stochastic Logic), an extension of the temporal logic CTL.

Definition 13. *The syntax of CSL is as follows:*

$$\Phi ::= \mathtt{true} \mid a \mid \neg\Phi \mid \Phi \wedge \Phi \mid \mathtt{P}_{\sim p}[\phi] \mid \mathtt{S}_{\sim p}[\Phi]$$
$$\phi ::= \mathtt{X}\ \Phi \mid \Phi\ \mathtt{U}^I\ \Phi$$

where a is an atomic proposition, $\sim \in \{<, \leqslant, \geqslant, >\}$, $p \in [0,1]$ and I is an interval of $\mathbb{R}_{\geqslant 0}$.

As for PCTL, $\mathtt{P}_{\sim p}[\phi]$ indicates that the probability of the path formula ϕ being satisfied from a given state meets the bound $\sim p$. Path formulae are the same for CSL as for PCTL, except that the parameter of the 'until' operator is an interval I of the non-negative reals, rather than simply an integer upper bound. The path formula $\Phi\ \mathtt{U}^I\ \Psi$ holds if Ψ is satisfied at some time instant in the interval I and Φ holds at all preceding time instants. To avoid confusion, we will refer to this as the 'time-bounded until' operator. Similarly to PCTL, the standard 'unbounded until' operator can be derived by considering the interval $I = [0, \infty)$. The $\mathtt{S}$ operator describes the steady-state behaviour of the CTMC.

The formula $\mathtt{S}_{\sim p}[\Phi]$ asserts that the steady-state probability of being in a state satisfying Φ meets the bound $\sim p$.

As with PCTL, we write $s \models \Phi$ to indicate that a CSL formula Φ is satisfied in a state s and denote by $Sat(\Phi)$ the set $\{s \in S \mid s \models \Phi\}$. Similarly, for a path formula ϕ satisfied by path ω, we write $\omega \models \phi$. The semantics of CSL over CTMCs is defined as follows.

Definition 14. *Let $\mathcal{C} = (S, \bar{s}, \mathbf{R}, L)$ be a labelled CTMC. For any state $s \in S$ the relation $s \models \Phi$ is defined inductively by:*

$$\begin{array}{rcl} s \models \mathtt{true} & \textit{for all } s \in S & \\ s \models a & \Leftrightarrow & a \in L(s) \\ s \models \neg\Phi & \Leftrightarrow & s \not\models \Phi \\ s \models \Phi \wedge \Psi & \Leftrightarrow & s \models \Phi \wedge s \models \Psi \\ s \models \mathtt{P}_{\sim p}[\phi] & \Leftrightarrow & Prob^{\mathcal{C}}(s, \phi) \sim p \\ s \models \mathtt{S}_{\sim p}[\Phi] & \Leftrightarrow & \sum_{s' \models \Phi} \pi^{\mathcal{C}}_s(s') \sim p \end{array}$$

where:

$$Prob^{\mathcal{C}}(s, \phi) \stackrel{\text{def}}{=} Pr_s\{\omega \in Path^{\mathcal{C}}(s) \mid \omega \models \phi\}$$

and for any path $\omega \in Path^{\mathcal{C}}(s)$:

$$\begin{array}{l} \omega \models \mathtt{X}\ \Phi \quad \Leftrightarrow \omega(1) \textit{ is defined and } \omega(1) \models \Phi \\ \omega \models \Phi\ \mathtt{U}^I\ \Psi \Leftrightarrow \exists t \in I.\, (\omega@t \models \Psi \wedge \forall x \in [0, t).\, (\omega@x \models \Phi))\,. \end{array}$$

As discussed in [12], for any path formula Φ, the set $\{\omega \in Path^{\mathcal{C}}(s) \mid \omega \models \phi\}$ is a measurable set of $(Path^{\mathcal{C}}(s), \Sigma_{Path^{\mathcal{C}}(s)})$, and hence Pr_s is well defined over this set. In addition the steady-state probabilities $\pi^{\mathcal{C}}_s(s')$ always exists as $\mathcal{C}$ contains finitely many states [59].

As with PCTL, we can easily derive CSL operators for $\mathtt{false}$, $\vee$ and $\rightarrow$. Similarly, we can use the $\Diamond$ and $\Box$ temporal operators:

$$\begin{array}{l} \mathtt{P}_{\sim p}[\Diamond^I \Phi] \equiv \mathtt{P}_{\sim p}[\mathtt{true}\ \mathtt{U}^I\ \Phi] \\ \mathtt{P}_{\sim p}[\Box^I \Phi] \equiv \mathtt{P}_{\tilde{\sim} 1-p}[\Diamond^I \neg\Phi]\,. \end{array}$$

It is also worth noting that, despite the fact that CSL does not explicitly include operators to reason about transient probabilities, the following can be used to reason about the probability of satisfying a formula Φ at time instant t:

$$\mathtt{P}_{\sim p}[\Diamond^{[t,t]} \Phi]\,.$$

Example 14. Below are some typical examples of CSL formulae:

- $\mathtt{P}_{>0.9}[\Diamond^{[0,4.5]}\ \mathit{served}]$ - the probability that a request is served within the first 4.5 seconds is greater than 0.9;
- $\mathtt{P}_{\leqslant 0.1}[\Diamond^{[10,\infty)}\ \mathit{error}]$ - the probability that an error occurs after 10 seconds of operation is at most 0.1;

– $\mathit{down} \rightarrow \mathtt{P}_{>0.75}[\neg \mathit{fail}\ \mathtt{U}^{[1,2]}\ \mathit{up}\ ]$ - when a shutdown occurs, the probability of system recovery being completed in between 1 and 2 hours without further failures occurring is greater than 0.75;
– $\mathtt{S}_{<0.01}[\mathit{insufficient\_routers}]$ - in the long-run, the probability that an inadequate number of routers are operational is less than 0.01.

4.4 CSL Model Checking

In this section we consider a model checking algorithm for CSL over CTMCs. CSL model checking was shown to be decidable (for rational time bounds) in [4] and a model checking algorithm first presented in [12]. We use these techniques plus the subsequent improvements made in [10,40].

The inputs to the algorithm are a labelled CTMC $\mathcal{C} = (S, \bar{s}, \mathbf{R}, L)$ and a CSL formula Φ. The output is the set of states $\mathit{Sat}(\Phi) = \{s \in S \mid s \models \Phi\}$. As for DTMCs and PCTL, we first construct the parse tree of the formula Φ and, working upwards towards the root of the tree, we recursively compute the set of states satisfying each subformula. By the end, we have determined whether each state in the model satisfies Φ. The algorithm for CSL operators can be summarised as follows:

$$\begin{aligned}
\mathit{Sat}(\mathtt{true}) &= S \\
\mathit{Sat}(a) &= \{s \mid a \in L(s)\} \\
\mathit{Sat}(\neg \Phi) &= S \backslash \mathit{Sat}(\Phi) \\
\mathit{Sat}(\Phi \wedge \Psi) &= \mathit{Sat}(\Phi) \cap \mathit{Sat}(\Psi) \\
\mathit{Sat}(\mathtt{P}_{\sim p}[\phi]) &= \{s \in S \mid \mathit{Prob}^{\mathcal{C}}(s, \phi) \sim p\} \\
\mathit{Sat}(\mathtt{S}_{\sim p}[\Phi]) &= \{s \in S \mid \textstyle\sum_{s' \models \Phi} \pi^{\mathcal{C}}_s(s') \sim p\}\,.
\end{aligned}$$

Model checking for the majority of these operators is trivial to implement and is, in fact, the same as for the non-probabilistic logic CTL. The exceptions are the $\mathtt{P}_{\sim p}[\cdot]$ and $\mathtt{S}_{\sim p}[\cdot]$ operators which are considered below.

$\mathtt{P}_{\sim p}[\mathtt{X}\ \Phi]$ formulae. The CSL 'next' operator is defined as for the PCTL equivalent. Furthermore, the definition does not relate to any of the real-time aspects of CTMCs: it depends only on the probability of moving to the next immediate state, and hence this operator can actually be model checked by using the PCTL algorithms of Section 3 on the embedded DTMC $\mathit{emb}(\mathcal{C})$ (see Definition 10).

Example 15. Consider the CTMC $\mathcal{C}_1$ in Fig. 4 and the CSL formula $\mathtt{P}_{\geqslant 0.5}[\mathtt{X}\ \mathit{full}]$. Working with the embedded DTMC $\mathit{emb}(\mathcal{C}_1)$ and following the algorithm of Section 3, we multiply the matrix $\mathbf{P}^{\mathit{emb}(\mathcal{C}_1)}$, given in Example 12, by the vector $(0, 0, 0, 1)$, yielding the probabilities $\underline{\mathit{Prob}}^{\mathcal{C}_1}(\mathtt{X}\ \mathit{full}) = (0, 0, \frac{1}{3}, 0)$. Hence, the formula is not true in any state of the CTMC.

$\mathtt{P}_{\sim p}[\Phi\ \mathtt{U}^I\ \Psi]$ formulae. For this operator, we need to determine the probabilities $\mathit{Prob}^{\mathcal{C}}(s, \Phi\ \mathtt{U}^I\ \Psi)$ for all states s where I is an arbitrary interval of the non-negative real numbers. Noting that:

$$\mathit{Prob}^{\mathcal{C}}(s, \Phi\ \mathtt{U}^I\ \Psi) = \mathit{Prob}^{\mathcal{C}}(s, \Phi\ \mathtt{U}^{cl(I)}\ \Psi)$$

where $cl(I)$ is the closure of the interval I, and that:

$$Prob^{\mathcal{C}}(s, \Phi\ \mathtt{U}^{[0,\infty)}\ \Psi) = Prob^{emb(\mathcal{C})}(s, \Phi\ \mathtt{U}^{\leqslant\infty}\ \Psi)$$

we are left with the following three cases for the interval I:

- $I = [0, t]$ for some $t \in \mathbb{R}_{\geqslant 0}$;
- $I = [t, t']$ for some $t, t' \in \mathbb{R}_{\geqslant 0}$ such that $t \leqslant t'$;
- $I = [t, \infty)$ for some $t \in \mathbb{R}_{\geqslant 0}$.

Following the method presented in [10], we will now show that the probabilities $Prob^{\mathcal{C}}(s, \Phi\ \mathtt{U}^{I}\ \Psi)$ for these cases can be computed using variants of uniformisation (see Section 4.2).

The case when $I = [0, t]$. Computing the probabilities in this case reduces to determining the least solution of the following set of integral equations: $Prob^{\mathcal{C}}(s, \Phi\ \mathtt{U}^{[0,t]}\ \Psi)$ equals 1 if $s \in Sat(\Psi)$, 0 if $s \in Sat(\neg\Phi \wedge \neg\Psi)$ and

$$Prob^{\mathcal{C}}(s, \Phi\ \mathtt{U}^{[0,t]}\ \Psi) = \int_0^t \sum_{s' \in S} \mathbf{P}^{\mathrm{emb}(\mathcal{C})}(s, s') \cdot E(s) \cdot e^{-E(s)\cdot x} \cdot Prob^{\mathcal{C}}(s', \Phi\ \mathtt{U}^{[0,t-x]}\ \Psi)\, dx$$

otherwise. Here, $E(s) \cdot e^{-E(s)\cdot x}$ denotes the probability density of taking some outgoing transition from s at time x. Note the resemblance with equations (1) for the PCTL bounded until operator. Originally, [12] proposed to do this via approximate solution of Volterra integral equation systems. Experiments in [34] showed that this method was generally slow and, in [8], a simpler alternative was presented which reduces the problem to transient analysis. This approach is outlined below.

Definition 15. *For any CTMC* $\mathcal{C} = (S, \bar{s}, \mathbf{R}, L)$ *and CSL formula* Φ*, let CTMC* $\mathcal{C}[\Phi] = (S, \bar{s}, \mathbf{R}[\Phi], L)$ *with* $\mathbf{R}[\Phi](s, s') = \mathbf{R}(s, s')$ *if* $s \not\models \Phi$ *and 0 otherwise.*

Note that, using Definition 8, we have that $emb(\mathcal{C}[\Phi]) = emb(\mathcal{C})[\Phi]$.

Proposition 3 ([8]). *For a CTMC* $\mathcal{C} = (S, \bar{s}, \mathbf{R}, L)$*, CSL formulae* Φ *and* Ψ *and positive real* $t \in \mathbb{R}_{\geqslant 0}$*:*

$$Prob^{\mathcal{C}}(s, \Phi\ \mathtt{U}^{[0,t]}\ \Psi) = \sum_{s' \models \Psi} \pi^{\mathcal{C}[\neg\Phi \vee \Psi]}_{s,t}(s')\,.$$

Consider the CTMC $\mathcal{C}[\neg\Phi \wedge \neg\Psi][\Psi] = \mathcal{C}[\neg\Phi \vee \Psi]$. Since a path in this CTMC cannot exit a state satisfying Ψ once it reaches one, and will never be able to reach a state satisfying Ψ if it enters one satisfying $\neg\Phi \wedge \neg\Psi$, the probability of the path formula $\Phi\ \mathtt{U}^{[0,t]}\ \Psi$ being satisfied in CTMC $\mathcal{C}$ is equivalent to the transient probability of being in a state satisfying Φ at time t in CTMC $\mathcal{C}[\neg\Phi \vee \Psi]$.

As shown in [40], uniformisation can be adapted to compute the vector of probabilities $\underline{Prob}^{\mathcal{C}}(\Phi\ \mathtt{U}^{[0,t]}\ \Psi)$ without having to resort to computing the probabilities for each state separately. More precisely, from Theorem 3:

$$\begin{aligned}\underline{Prob}^{\mathcal{C}}(\Phi\ \mathtt{U}^{[0,t]}\ \Psi) &= \boldsymbol{\Pi}_t^{\mathcal{C}[\neg\Phi\vee\Psi]}\cdot\underline{\Psi}\\ &= \left(\sum_{i=0}^{\infty}\gamma_{i,q\cdot t}\cdot\left(\mathbf{P}^{unif(\mathcal{C}[\neg\Phi\vee\Psi])}\right)^i\right)\cdot\underline{\Psi} \qquad \text{by (2)}\\ &= \sum_{i=0}^{\infty}\left(\gamma_{i,q\cdot t}\cdot\left(\mathbf{P}^{unif(\mathcal{C}[\neg\Phi\vee\Psi])}\right)^i\cdot\underline{\Psi}\right) \qquad \text{rearranging}\end{aligned}$$

Note that the inclusion of the vector $\underline{\Psi}$ within the brackets is vital since, like with uniformisation, it allows explicit computation of matrix powers to be avoided. Instead, each product is calculated as:

$$\left(\mathbf{P}^{unif(\mathcal{C})}\right)^0\cdot\underline{\Psi}=\boldsymbol{\Psi}\ \text{ and }\ \left(\mathbf{P}^{unif(\mathcal{C})}\right)^{i+1}\cdot\underline{\Psi}=\mathbf{P}^{unif(\mathcal{C})}\cdot\left(\left(\mathbf{P}^{unif(\mathcal{C})}\right)^i\cdot\underline{\Psi}\right),$$

reusing the computation from the previous iteration. As explained in Section 4.2, the infinite summation can be truncated using the techniques of Fox and Glynn [27]. In fact the summation can be truncated even sooner if the vector converges. As an additional optimisation, we can reuse the Prob0 algorithm, from Fig. 3 in Section 3, to initially identify all states s for which the probability $Prob^{\mathcal{C}}(s,\Phi\ \mathtt{U}^{[0,t]}\ \Psi)$ is 0.

Example 16. Consider the CTMC $\mathcal{C}_1$ in Fig. 4 and the CSL 'time-bounded until' formula $\mathtt{P}_{>0.65}[\mathtt{true}\ \mathtt{U}^{[0,7.5]}\ \mathit{full}]$. To compute the vector of probabilities $\underline{Prob}^{\mathcal{C}}(\mathtt{true}\ \mathtt{U}^{[0,7.5]}\ \mathit{full})$, i.e. the probability from each state that a state satisfying atomic proposition *full* is reached within 7.5 time units, we follow the procedure outlined above. First, observe that only state s_3 satisfies *full* and no states satisfy $\neg\mathtt{true}$. Hence, the only difference in the modified CTMC $\mathcal{C}_1[\neg\mathtt{true}\vee\mathit{full}]$ is that state s_3 made absorbing, i.e. the transition between states s_3 and s_2 is removed. Using the uniformisation rate $q = 4.5(=\max_{0\leqslant i\leqslant 3}E(s_i))$, the transition probability matrix for the uniformised DTMC of this modified CTMC $\mathcal{C}_1[\neg\mathtt{true}\vee\mathit{full}]$ is given by:

$$\mathbf{P}^{unif(\mathcal{C}_1[\neg\mathtt{true}\vee\mathit{full}])}=\begin{pmatrix}\frac{2}{3} & \frac{1}{3} & 0 & 0\\ \frac{2}{3} & 0 & \frac{1}{3} & 0\\ 0 & \frac{2}{3} & 0 & \frac{1}{3}\\ 0 & 0 & 0 & 1\end{pmatrix}.$$

Computing the summation of matrix-vector multiplications described above yields the solution:

$$\underline{Prob}^{\mathcal{C}_1}(\mathtt{true}\ \mathtt{U}^{[0,7.5]}\ \mathit{full})\approx(0.6482,0.6823,0.7811,1)$$

and we conclude that the CSL property is satisfied in states s_1, s_2 and s_3.

The case $I=[t,t']$. For this case, we split the computation into two parts. As shown in [8], we can consider separately the probabilities of: (a) staying in states satisfying Φ up until time t; (b) reaching a state satisfying Ψ, while remaining

in states satisfying Φ, within time $t' - t$. For the former, we use a similar idea to that used in the case when $I = [0, t]$, computing transient probabilities in a CTMC for which states satisfying $\neg\Phi$ have been made absorbing. We have:

$$\begin{aligned} Prob^{\mathcal{C}}(s, \Phi\ \mathtt{U}^{[t,t']}\ \Psi) &= \sum_{s' \models \Phi} \pi^{\mathcal{C}[\neg\Phi]}_{s,t}(s') \cdot Prob^{\mathcal{C}}(s', \Phi\ \mathtt{U}^{[0,t'-t]}\ \Psi) \\ &= \underline{\pi}^{\mathcal{C}[\neg\Phi]}_{s,t} \cdot \underline{Prob}^{\mathcal{C}}_{\Phi}(\Phi\ \mathtt{U}^{[0,t'-t]}\ \Psi) \end{aligned}$$

where $\underline{Prob}^{\mathcal{C}}_{\Phi}(\Phi\ \mathtt{U}^{[0,t'-t]}\ \Psi)$ is a vector with:

$$Prob^{\mathcal{C}}_{\Phi}(s, \Phi\ \mathtt{U}^{[0,t'-t]}\ \Psi) = \begin{cases} Prob^{\mathcal{C}}(s, \Phi\ \mathtt{U}^{[0,t'-t]}\ \Psi) & \text{if } s \models \Phi \\ 0 & \text{otherwise} \end{cases}$$

which can be computed using the method described above. The overall computation can be performed as a summation, in the style of uniformisation, to determine the probability for all states at once:

$$\begin{aligned} \underline{Prob}^{\mathcal{C}}(\Phi\ \mathtt{U}^{[t,t']}\ \Psi) &= \boldsymbol{\Pi}^{\mathcal{C}[\neg\Phi]}_{t} \cdot \underline{Prob}^{\mathcal{C}}_{\Phi}(\Phi\ \mathtt{U}^{[0,t'-t]}\ \Psi) \\ &= \left(\sum_{i=0}^{\infty} \gamma_{i,q\cdot t} \cdot \left(\mathbf{P}^{unif(\mathcal{C}[\neg\Phi])}\right)^i\right) \cdot \underline{Prob}^{\mathcal{C}}_{\Phi}(\Phi\ \mathtt{U}^{[0,t'-t]}\ \Psi) \\ &= \sum_{i=0}^{\infty} \left(\gamma_{i,q\cdot t} \cdot \left(\mathbf{P}^{unif(\mathcal{C}[\neg\Phi])}\right)^i \cdot \underline{Prob}^{\mathcal{C}}_{\Phi}(\Phi\ \mathtt{U}^{[0,t'-t]}\ \Psi)\right) \end{aligned}$$

Again, this summation can be truncated and performed using only scalar and matrix-vector multiplication.

The case $I = [t, \infty)$. This case is, in fact, almost identical to the previous one. We again split the computation into two parts. Here, however, the second part is an unbounded, rather than time-bounded, 'until' formula, and hence the embedded DTMC can be used in this case. More precisely, we have:

$$Prob^{\mathcal{C}}(s, \Phi\ \mathtt{U}^{[t,\infty)}\ \Psi) = \underline{\pi}^{\mathcal{C}[\neg\Phi]}_{s,t} \cdot \underline{Prob}^{\mathcal{C}}_{\Phi}(\Phi\ \mathtt{U}\ \Psi) = \underline{\pi}^{\mathcal{C}[\neg\Phi]}_{s,t} \cdot \underline{Prob}^{emb(\mathcal{C})}_{\Phi}(\Phi\ \mathtt{U}\ \Psi)\,.$$

Similarly to the above, this can be compute for all states:

$$\underline{Prob}^{\mathcal{C}}(\Phi\ \mathtt{U}^{[t,\infty)}\ \Psi) = \sum_{i=0}^{\infty} \left(\gamma_{i,q\cdot t} \cdot \left(\mathbf{P}^{unif(\mathcal{C}[\neg\Phi])}\right)^i \cdot \underline{Prob}^{emb(\mathcal{C})}_{\Phi}(\Phi\ \mathtt{U}\ \Psi)\right)\,.$$

$\mathtt{S}_{\sim p}[\Phi]$ formulae. A state s satisfies the formula $\mathtt{S}_{\sim p}[\Phi]$ if $\sum_{s' \models \Phi} \pi^{\mathcal{C}}_{s}(s') \sim p$. Therefore, to model check the formula $\mathtt{S}_{\sim p}[\Phi]$, we must compute the steady-state probabilities $\pi^{\mathcal{C}}_{s}(s')$ for all states s and s'. We first consider the simple case when $\mathcal{C}$ is irreducible.

The case when $\mathcal{C}$ is irreducible. As described in Section 4.2, the steady-state probabilities of $\mathcal{C}$ are independent of the starting state, and therefore we denote

by $\pi^{\mathcal{C}}(s)$ and $\underline{\pi}^{\mathcal{C}}$ the steady-state probability of being in the state s and the vector of all such probabilities, respectively. These probabilities can be computed as the unique solution of the linear equation system:

$$\underline{\pi}^{\mathcal{C}} \cdot \mathbf{Q} = \underline{0} \;\text{ and }\; \textstyle\sum_{s \in S} \pi^{\mathcal{C}}(s) = 1 \,. \tag{3}$$

This system can be solved by any standard approach, for example using direct methods, such as Gaussian elimination, or iterative methods, such as Jacobi and Gauss-Seidel. The satisfaction of the CSL formula, which in this case will be the same for all states, can be determined by summing the steady-state probabilities for all states satisfying Φ and comparing this result to the bound in the formula. More precisely, for any state $s \in S$:

$$s \models \mathtt{S}_{\sim p}[\Phi] \quad \Leftrightarrow \quad \textstyle\sum_{s' \models \Phi} \pi^{\mathcal{C}}(s') \sim p \,.$$

The case when $\mathcal{C}$ is reducible. In this case the procedure is more complex. First graph analysis is carried out to determine the set $bscc(\mathcal{C})$ of bottom strongly connected components (BSCCs) of $\mathcal{C}$, i.e. the set of strongly connect components of $\mathcal{C}$ that, once entered, cannot be left any more. Each individual BSCC $\mathcal{B} \in bscc(\mathcal{C})$ can be treated as an irreducible CTMC, and hence the steady-state probability distribution $\underline{\pi}^{\mathcal{B}}$ can be determined using the method described in the previous case.

Next, we calculate the probability of reaching each BSCC $\mathcal{B} \in bscc(\mathcal{C})$ from each state s of $\mathcal{C}$. In fact, this is simply $Prob^{emb(\mathcal{C})}(s, \Diamond\, a_{\mathcal{B}})$, where $a_{\mathcal{B}}$ is an atomic proposition true only in the states $s' \in \mathcal{B}$. Then, for states $s, s' \in S$, the steady-state probability $\pi^{\mathcal{C}}_s(s')$ can be computed as:

$$\pi^{\mathcal{C}}_s(s') = \begin{cases} Prob^{emb(\mathcal{C})}(s, \Diamond\, a_{\mathcal{B}}) \cdot \pi^{\mathcal{B}}(s') & \text{if } s' \in \mathcal{B} \text{ for some } \mathcal{B} \in bscc(\mathcal{C}) \\ 0 & \text{otherwise.} \end{cases}$$

Note that, since the steady-state probabilities $\pi^{\mathcal{B}}(s')$ are independent of s, the total work required to compute $\pi^{\mathcal{C}}_s(s')$ for all $s, s' \in S$ is the solution of two linear equation systems for each BSCC in the CTMC: one to obtain the vector of probabilities $\underline{\pi}^{\mathcal{B}}$ and another for the vector of probabilities $\underline{Prob}^{emb(\mathcal{C})}(\Diamond\, a_{\mathcal{B}})$. Computation of the BSCCs in the CTMC requires an analysis of its underlying graph structure and can be performed using classic algorithms based on depth-first search [60].

Example 17. Consider the CTMC $\mathcal{C}_1$ in Fig. 4 and the CSL 'steady-state' formula $\mathtt{S}_{<0.1}[\mathit{full}]$. From inspection, we see that the CTMC comprises a single BSCC containing all 4 states. Hence, the steady-state probabilities are computed by solving the linear equation system:

$$\begin{aligned}
-\tfrac{3}{2} \cdot \pi^{\mathcal{C}_1}(s_0) + 3 \cdot \pi^{\mathcal{C}_1}(s_1) &= 0 \\
\tfrac{3}{2} \cdot \pi^{\mathcal{C}_1}(s_0) - \tfrac{9}{2} \cdot \pi^{\mathcal{C}_1}(s_1) + 3 \cdot \pi^{\mathcal{C}_1}(s_2) &= 0 \\
\tfrac{3}{2} \cdot \pi^{\mathcal{C}_1}(s_1) - \tfrac{9}{2} \cdot \pi^{\mathcal{C}_1}(s_2) + 3 \cdot \pi^{\mathcal{C}_1}(s_3) &= 0 \\
\tfrac{3}{2} \cdot \pi^{\mathcal{C}_1}(s_2) - 3 \cdot \pi^{\mathcal{C}_1}(s_3) &= 0 \\
\pi^{\mathcal{C}_1}(s_0) + \pi^{\mathcal{C}_1}(s_1) + \pi^{\mathcal{C}_1}(s_2) + \pi^{\mathcal{C}_1}(s_3) &= 1
\end{aligned}$$

which has the solution $\underline{\pi}^{\mathcal{C}_1} = (\frac{8}{15}, \frac{4}{15}, \frac{2}{15}, \frac{1}{15})$. State s_3 is the only state satisfying atomic proposition *full*, and thus the CSL formula is true in all states.

4.5 Extending CTMCs and CSL with Rewards

As for DTMCs, given a CTMC $\mathcal{C} = (S, \bar{s}, \mathbf{R}, L)$, we can enrich $\mathcal{C}$ with a reward structure $(\underline{\rho}, \boldsymbol{\iota})$. Recall that the state reward function $\underline{\rho} : S \to \mathbb{R}_{\geqslant 0}$ defines this as the rate at which reward is acquired in a state and the transition reward function $\boldsymbol{\iota} : S \times S \to \mathbb{R}_{\geqslant 0}$ defines the reward acquired each time a transition occurs. Note that, since we are now in the continuous time setting, a reward of $t \cdot \underline{\rho}(s)$ will be acquired if the CTMC remains in state s for $t \in \mathbb{R}_{\geqslant 0}$ time units.

Example 18. Returning to the CTMC $\mathcal{C}_1$ of Example 12 (see Fig. 4), we consider two different reward structures:

– $(\underline{0}, \boldsymbol{\iota}^{\mathcal{C}_1})$ where $\boldsymbol{\iota}^{\mathcal{C}_1}$ assigns 1 to the transitions corresponding to a request being served and 0 to all other transitions, that is:

$$\boldsymbol{\iota}^{\mathcal{C}_1} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}.$$

Such a structure can be used for measures relating to the number of requests served within a given time interval or in the long-run.

– $(\underline{\rho}^{\mathcal{C}_1}, \mathbf{0})$ where $\underline{\rho}^{\mathcal{C}_1}$ associates with each state the number of requests that are awaiting service:

$$\underline{\rho}^{\mathcal{C}_1} = (0, 1, 2, 3)\,.$$

This structure is used when one is interested in the queue size at any time instant or in the long-run.

The construction of both the embedded DTMC $emb(\mathcal{C})$ (see Definition 10) and uniformised DTMC $unif(\mathcal{C})$ (see Definition 12) can be extended to incorporate the reward structure $(\boldsymbol{\iota}, \underline{\rho})$. In both constructions the transition reward function does not change, that is, $\boldsymbol{\iota}^{emb(\mathcal{C})} = \boldsymbol{\iota}^{unif(\mathcal{C})} = \boldsymbol{\iota}$. On the other hand, the constructed state reward function takes into account the expected time that the CTMC remains in each state. More precisely, if q is the uniformisation rate used in the construction of the uniformised DTMC, then for any $s \in S$:

$$\underline{\rho}^{emb(\mathcal{C})}(s) = E(s) \cdot \underline{\rho}(s) \;\text{ and }\; \underline{\rho}^{unif(\mathcal{C})}(s) = \tfrac{1}{q} \cdot \underline{\rho}(s)\,.$$

We extend the syntax of logic CSL to allow for specifications relating to rewards by introducing the following formulae:

$$\mathtt{R}_{\sim r}[\mathtt{C}^{\leqslant t}] \;\big|\; \mathtt{R}_{\sim r}[\mathtt{I}^{=t}] \;\big|\; \mathtt{R}_{\sim r}[\mathtt{F}\ \Phi] \;\big|\; \mathtt{R}_{\sim r}[\mathtt{S}]$$

where $\sim \in \{<, \leqslant, \geqslant, >\}$, $r, t \in \mathbb{R}_{\geqslant 0}$ and Φ is a CSL formula. Intuitively, a state s satisfies $\mathtt{R}_{\sim r}[\mathtt{C}^{\leqslant t}]$ if, from state s, the expected reward *cumulated* up until t time

units have elapsed satisfies $\sim r$; $\mathtt{R}_{\sim r}[\mathtt{I}^{=t}]$ is true if, from state s, the expected state reward at time instant t meets the bound $\sim r$; $\mathtt{R}_{\sim r}[\mathtt{F}\ \Phi]$ is true if, from state s, the expected reward cumulated before a state satisfying Φ is reached meets the bound $\sim r$; and $\mathtt{R}_{\sim r}[\mathtt{S}]$ is true if, from state s, the long-run average expected reward satisfies $\sim r$.

Formally, given a CTMC $\mathcal{D} = (S, \bar{s}, \mathbf{R}, L)$, the semantics of these formulae is defined as follows. For any $s \in S$, $r, t \in \mathbb{R}_{\geqslant 0}$ and PCTL formula Φ:

$$\begin{array}{rcl} s \models \mathtt{R}_{\sim r}[\mathtt{C}^{\leqslant t}] & \Leftrightarrow & \mathit{Exp}^{\mathcal{C}}(s, X_{\mathtt{C}^{\leqslant t}}) \sim r \\ s \models \mathtt{R}_{\sim r}[\mathtt{I}^{=t}] & \Leftrightarrow & \mathit{Exp}^{\mathcal{C}}(s, X_{\mathtt{I}^{=t}}) \sim r \\ s \models \mathtt{R}_{\sim r}[\mathtt{F}\ \Phi] & \Leftrightarrow & \mathit{Exp}^{\mathcal{C}}(s, X_{\mathtt{F}\Phi}) \sim r \\ s \models \mathtt{R}_{\sim r}[\mathtt{S}] & \Leftrightarrow & \lim_{t\to\infty} \frac{1}{t} \cdot \mathit{Exp}^{\mathcal{C}}(s, X_{\mathtt{C}^{\leqslant t}}) \sim r \end{array}$$

where $\mathit{Exp}^{\mathcal{C}}(s, X)$ denotes the expectation of the random variable X with respect to the probability measure Pr_s and for any path $\omega = s_0 t_0 s_1 t_1 s_2 \cdots \in \mathit{Path}^{\mathcal{C}}(s)$:

$$X_{\mathtt{C}^{\leqslant t}}(\omega) \stackrel{\text{def}}{=} \sum_{i=0}^{j_t-1} \left(t_i \cdot \underline{\rho}(s_i) + \boldsymbol{\iota}(s_i, s_{i+1})\right) + \left(t - \sum_{i=0}^{j_t-1} t_i\right) \cdot \underline{\rho}(s_{j_t})$$

$$X_{\mathtt{I}^{=t}}(\omega) \stackrel{\text{def}}{=} \underline{\rho}(\omega@t)$$

$$X_{\mathtt{F}\Phi}(\omega) \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } \omega(0) \models \Phi \\ \infty & \text{if } \forall i \in \mathbb{N}.\ s_i \not\models \Phi \\ \sum_{i=0}^{\min\{j \mid s_j \models \Phi\}-1} t_i \cdot \underline{\rho}(s_i) + \boldsymbol{\iota}(s_i, s_{i+1}) & \text{otherwise} \end{cases}$$

and $j_t = \min\{j \mid \sum_{i=0}^{j} t_i \geqslant t\}$.

Example 19. Below are some typical examples of reward based formulae:

- $\mathtt{R}_{>3.6}[\mathtt{C}^{\leqslant 4.5}]$ - the expected number of requests served within the first 4.5 seconds of operations is greater than 3.6;
- $\mathtt{R}_{<2}[\mathtt{I}^{=6.7}]$ - the expected size of the queue when 6.7 time units have elapsed is less than 2;
- $\mathtt{R}_{<10}[\mathtt{F}\ \mathit{full}]$ - the expected number of requests served before the queue becomes full is less than 10;
- $\mathtt{R}_{\geqslant 1.2}[\mathtt{S}]$ - the expected long-run queue size is at least 1.2.

We now consider the computation of the expected values of the different random variables defined above.

The random variable $X_{\mathtt{C}^{\leqslant t}}$. Based on the results in [43,44], we can use the uniformised DTMC to compute the expectation of this random variable. More precisely, we have have the following result.

Proposition 4 ([44]). *For a CTMC $\mathcal{C} = (S, \bar{s}, \mathbf{R}, L)$, state $s \in S$ and positive real $t \in \mathbb{R}_{\geqslant 0}$:*

$$\mathit{Exp}^{\mathcal{C}}(s, X_{\mathtt{C}^{\leqslant t}}) \ = \ \sum_{i=0}^{\infty} \bar{\gamma}_{i,q\cdot t} \cdot \left(\mathbf{P}^{\mathit{unif}(\mathcal{C})}\right)^i \cdot \underline{f_q}(\underline{\rho}, \boldsymbol{\iota})$$

where

$$\bar{\gamma}_{i,q\cdot t} \stackrel{\text{def}}{=} \int_0^t \gamma_{i,q\cdot u}\, du \;=\; \frac{1}{q}\sum_{j=i+1}^{\infty}\gamma_{j,q\cdot t} \;=\; \frac{1}{q}\left(1-\sum_{j=i}^{i}\gamma_{j,q\cdot t}\right),$$

$\underline{\pi}^{unif(\mathcal{C})}_{s,i}$ *denotes the probability distribution in* $unif(\mathcal{C})$ *after* i *steps when starting in* s, q *is the uniformisation rate and*

$$\underline{f_q}(\underline{\rho},\boldsymbol{\iota}) = \underline{\rho} + q\cdot\left(\mathbf{P}^{unif(\mathcal{C})}\bullet\boldsymbol{\iota}\right)\cdot\underline{1}$$

with $\bullet$ *denoting the Schur or entry-wise multiplication of matrices and* $\underline{1}$ *a vector with all entries equal to 1.*

Similarly to computing the vector of probabilities $\underline{Prob}^{\mathcal{C}}(\Phi\ \mathtt{U}^{[0,t]}\ \Psi)$, we can both truncate the summation and use only scalar and matrix-vector multiplication in the computation. In this case, to compute the coefficients $\bar{\gamma}_{i,q\cdot t}$, we can employ the method (based on Fox and Glynn [27]) given in [42].

Example 20. Returning to the CTMC $\mathcal{C}_1$ of Example 12 and the reward structure $(\mathbf{0},\underline{\rho}^{\mathcal{C}_1})$ of Example 18, the expected number of requests served after 5.5 time units have elapsed is given by:

$$\underline{Exp}^{\mathcal{C}_1}(X_{\mathtt{C}^{\leqslant 5.5}}) \approx (7.0690, 8.0022, 8.8020, 9.3350)$$

and hence only state s_3 satisfies $\mathtt{R}_{>9}[\mathtt{C}^{\leqslant 5.5}]$.

The random variable $X_{\mathtt{I}^{=t}}$. In this case, using the fact that:

$$Exp^{\mathcal{C}}(s, X_{\mathtt{I}^{=t}}) = \sum_{s'\in S}\underline{\rho}(s')\cdot\underline{\pi}^{\mathcal{C}}_{s,t}(s')$$

we can again use the uniformised DTMC $unif(\mathcal{C})$ to compute the expectation. More precisely, we can compute the vector $\underline{Exp}^{\mathcal{C}}(X_{\mathtt{I}^{=t}})$ through the following sum over vectors of coefficients:

$$\underline{Exp}^{\mathcal{C}}(X_{\mathtt{I}^{=t}}) = \sum_{i=0}^{\infty}\gamma_{i,q\cdot t}\cdot\left(\mathbf{P}^{unif(\mathcal{C})}\right)^i\cdot\underline{\rho}$$

which again can be truncated and computed using only scalar and matrix-vector multiplications.

Example 21. Returning to the CTMC $\mathcal{C}_1$ in Example 12 and the reward structure $(\underline{\rho}^{\mathcal{C}_1},\mathbf{0})$ of Example 18, the expected size of the queue after 1 time unit has elapsed is given by:

$$\underline{Exp}^{\mathcal{C}_1}(X_{\mathtt{I}^{=1}}) \approx (0.5929, 0.7352, 1.0140, 1.2875)$$

and hence all states satisfy the formula $\mathtt{R}_{<2}[\mathtt{I}^{=1}]$.

The random variable $X_{\mathtt{F}\Phi}$. To compute the expectations in this case, we use the fact that:

$$Exp^{\mathcal{C}}(s, X_{\mathtt{F}\Phi}) = Exp^{emb(\mathcal{C})}(s, X_{\mathtt{F}\Phi})$$

that is, we compute the expectations by constructing the embedded DTMC $emb(\mathcal{C})$ and employing the algorithms for verifying DTMCs against PCTL given in Section 3.3.

Example 22. Consider the CTMC $\mathcal{C}_1$ of Example 12, the reward structure $(\underline{0}, \boldsymbol{\iota}^{\mathcal{C}_1})$ of Example 18. The formula $\mathtt{R}_{<7}[\mathtt{F}\ \mathit{full}]$, in this case, states that the expected number of requests served before the queue becomes full is less than 7. Now, computing the expectations $Exp^{emb(\mathcal{C}_1)}(s, X_{\mathtt{F}\mathit{full}})$ according to Section 3.4:

$$\begin{aligned} Sat(\mathit{full}) &= \{s_3\} \\ Sat(\mathtt{P}_{<1}[\Diamond\ \mathit{full}]) &= S \setminus \textsc{Prob1}(S, Sat(\mathit{full}), \textsc{Prob0}(S, Sat(\mathit{full}))) \\ &= S \setminus \{s_0, s_1, s_2, s_3\} = \varnothing \end{aligned}$$

leading to the linear equation system:

$$\begin{aligned} Exp^{emb(\mathcal{C}_1)}(s_0, X_{\mathtt{F}\mathit{full}}) &= 1 \cdot Exp^{emb(\mathcal{C}_1)}(s_1, X_{\mathtt{F}\mathit{full}}) \\ Exp^{emb(\mathcal{C}_1)}(s_1, X_{\mathtt{F}\mathit{full}}) &= \tfrac{2}{3} \cdot \big(1 + Exp^{emb(\mathcal{C}_1)}(s_0, X_{\mathtt{F}\mathit{full}})\big) + \tfrac{1}{3} Exp^{emb(\mathcal{C}_1)}(s_2, X_{\mathtt{F}\mathit{full}}) \\ Exp^{emb(\mathcal{C}_1)}(s_2, X_{\mathtt{F}\mathit{full}}) &= \tfrac{2}{3} \cdot \big(1 + Exp^{emb(\mathcal{C}_1)}(s_1, X_{\mathtt{F}\mathit{full}})\big) \\ Exp^{emb(\mathcal{C}_1)}(s_3, X_{\mathtt{F}\mathit{full}}) &= 0\,. \end{aligned}$$

Solving this system of equations gives $\underline{Exp^{emb(\mathcal{C}_1)}(X_{\mathtt{F}\mathit{full}})} = (8, 8, 6, 0)$, and therefore, since $Exp^{\mathcal{C}_1}(s, X_{\mathtt{F}\mathit{full}}) = Exp^{emb(\mathcal{C}_1)}(s, X_{\mathtt{F}\mathit{full}})$, only states s_2 and s_3 satisfy the formula $\mathtt{R}_{\leqslant 7}[\mathtt{F}\ \mathit{full}]$.

The random variable $X_{\mathtt{S}}$. As in the case of the operator $\mathtt{S}_{\sim p}[\cdot]$, we consider the cases when $\mathcal{C}$ is irreducible and reducible separately.

The case when $\mathcal{C}$ is irreducible. If $\underline{\pi}^{\mathcal{C}}$ is the vector of the steady-state probabilities (recall that when $\mathcal{C}$ is irreducible the steady-state probabilities are independent of the starting state), we have:

$$Exp^{\mathcal{C}}(s, X_{\mathtt{S}}) = \underline{\pi}^{\mathcal{C}} \cdot \underline{\rho} + \underline{\pi}^{\mathcal{C}} \cdot (\mathbf{R} \bullet \boldsymbol{\iota}) \cdot \underline{1}$$

with $\bullet$ again denoting the Schur or entry-wise multiplication of matrices and $\underline{1}$ a vector with all entries equal to 1. Note that since the expectation is independent of the starting state, we denote the expectation by $Exp^{\mathcal{C}}(X_{\mathtt{S}})$. The computation in this case therefore requires the computation of the steady-state probabilities of $\mathcal{C}$, which reduces to solving the linear equation system given in (3).

The case when $\mathcal{C}$ is reducible. Similarly, to the approach for checking formulae of the form $\mathtt{S}_{\sim p}[\Phi]$, first, through graph analysis, we determine the set $bscc(\mathcal{C})$ of BSCCs of $\mathcal{C}$. Next, treating each individual $\mathcal{B} \in bscc(\mathcal{C})$ as an irreducible

CTMC, we compute the expectations $Exp^{\mathcal{B}_i}(X_{\mathtt{S}})$ and determine the vector of probabilities $\underline{Prob}^{emb(\mathcal{C})}(\Diamond\, a_{\mathcal{B}})$ for each $\mathcal{B} \in bscc(\mathcal{C})$. Finally, for each state $s \in S$:

$$Exp^{\mathcal{C}}(s, \mathtt{S}) = \sum_{\mathcal{B} \in bscc(\mathcal{C})} Prob^{\mathrm{emb}(\mathcal{C})}(s, \Diamond\, a_{\mathcal{B}}) \cdot Exp^{\mathcal{B}}(\mathtt{S})\ .$$

Example 23. Returning once again to the CTMC $\mathcal{C}_1$ in Example 12, using the steady-state probabilities computed earlier and the reward structure $(\underline{0}, \boldsymbol{\iota}^{\mathcal{C}_1})$ of Example 18, the long-run average expected number of requests served is given by:

$$\begin{aligned}
\underline{\pi}^{\mathcal{C}} \cdot \underline{0} + \underline{\pi}^{\mathcal{C}} \cdot (\mathbf{R}_1 \bullet \boldsymbol{\iota}^{\mathcal{C}_1}) \cdot \underline{1} &= (\tfrac{8}{15}, \tfrac{4}{15}, \tfrac{2}{15}, \tfrac{1}{15}) \cdot \left(\begin{pmatrix} 0 & \frac{3}{2} & 0 & 0 \\ 3 & 0 & \frac{3}{2} & 0 \\ 0 & 3 & 0 & \frac{3}{2} \\ 0 & 0 & 3 & 0 \end{pmatrix} \bullet \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \right) \cdot \underline{1} \\
&= (\tfrac{8}{15}, \tfrac{4}{15}, \tfrac{2}{15}, \tfrac{1}{15}) \cdot \begin{pmatrix} 0 & 0 & 0 & 0 \\ 3 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 \\ 0 & 0 & 3 & 0 \end{pmatrix} \cdot \underline{1} \\
&= (\tfrac{8}{15}, \tfrac{4}{15}, \tfrac{2}{15}, \tfrac{1}{15}) \cdot \begin{pmatrix} 0 \\ 3 \\ 3 \\ 3 \end{pmatrix} = \tfrac{7}{5}
\end{aligned}$$

and thus no states satisfy the formula $\mathtt{R}_{\geqslant 1.5}[\mathtt{S}]$ when the reward structure $(\underline{0}, \boldsymbol{\iota}^{\mathcal{C}_1})$ is associated with the CTMC $\mathcal{C}_1$.

On the other hand, using the reward structure $(\underline{\rho}^{\mathcal{C}_1}, \mathbf{0})$ of Example 18, the long-run average size of the queue is given by:

$$\underline{\pi}^{\mathcal{C}} \cdot \underline{c}_1 + \underline{\pi}^{\mathcal{C}} \cdot (\mathbf{R}_1 \bullet \mathbf{0}) \cdot \underline{1} = (\tfrac{8}{15}, \tfrac{4}{15}, \tfrac{2}{15}, \tfrac{1}{15}) \cdot \begin{pmatrix} 0 \\ 1 \\ 2 \\ 3 \end{pmatrix} = \tfrac{11}{15}$$

and hence all states satisfy the formula $\mathtt{R}_{\leqslant 0.8}[\mathtt{S}]$ when the reward structure $(\underline{\rho}^{\mathcal{C}_1}, \mathbf{0})$ is associated with the CTMC $\mathcal{C}_1$.

4.6 Complexity of CSL Model Checking

The overall time complexity for model checking a CSL formula Φ against a CTMC $\mathcal{C} = (S, \bar{s}, \mathbf{R}, L)$ is linear in $|\Phi|$, polynomial in $|S|$ and linear in $q \cdot t_{\max}$, where $q = \max_{s \in S} |\mathbf{Q}(s, s)|$ and $t_{\max}$ is the maximum value found in the parameter of a 'time-bounded until' operator. For formulae of the form $\mathtt{P}_{\sim p}[\Phi\ \mathtt{U}^{[0,\infty)}\ \Psi]$, $\mathtt{S}_{\sim p}[\Phi]$, $\mathtt{R}_{\sim r}[\mathtt{F}\ \Phi]$ and $\mathtt{R}_{\sim r}[\mathtt{S}]$ a solution of a linear equation system of size $|S|$ is required. This can be done with Gaussian elimination, the complexity of which

is cubic in the size of the system. For formula of the form $\mathtt{P}_{\sim p}[\Phi\ \mathtt{U}^I\ \Psi]$, $\mathtt{R}_{\sim r}[\mathtt{C}^{\leqslant t}]$ and $\mathtt{R}_{\sim r}[\mathtt{I}^{=t}]$ we must perform at most two iterative summations, each step of which requires a matrix-vector multiplication. This operation is quadratic in the size of the matrix, i.e. $|S|$. The total number of iterations required is determined by the upper bound supplied by the algorithm of Fox and Glynn [27], which for large $q{\cdot}t$ is linear in $q{\cdot}t$.

5 Stochastic Model Checking in Practice

In this section we first give a high-level overview of the functionality of the stochastic model checker PRISM and then discuss three case studies employing stochastic model checking and PRISM.

5.1 The Probabilistic Model Checker PRISM

PRISM [36,53] is a probabilistic model checker developed at the University of Birmingham. It accepts probabilistic models described in its *modelling language*, a simple, high-level state-based language. Three types of probabilistic models are supported directly; these are discrete-time Markov chains (DTMCs), Markov decision processes (MDPs), and continuous-time Markov chains (CTMCs). Markov decision processes, not considered in this tutorial, extend DTMCs by allowing non-deterministic behaviour that is needed, for example, to model asynchronous parallel composition. For a detailed introduction to model checking of MDPs see, for example, [56]. Additionally, probabilistic timed automata (PTAs) are partially supported, with the subset of diagonal-free PTAs supported directly via *digital clocks* [47]. Properties are specified using PCTL for DTMCs and MDPs, and CSL for CTMCs. Probabilistic timed automata have a logic PTCTL, an extension of TCTL, a subset of which is supported via a connection to the timed automata model checking tool Kronos [24].

Tool Overview. PRISM first parses the model description and constructs an internal representation of the probabilistic model, computing the reachable state space of the model and discarding any unreachable states. This represents the set of all feasible configurations which can arise in the modelled system. Next, the specification is parsed and appropriate model checking algorithms are performed on the model by induction over syntax. In some cases, such as for properties which include a probability bound, PRISM will simply report a true/false outcome, indicating whether or not each property is satisfied by the current model. More often, however, properties return *quantitative* results and PRISM reports, for example, the actual probability of a certain event occurring in the model. Furthermore, PRISM supports the notion of *experiments*, which is a way of automating multiple instances of model checking. This allows the user to easily obtain the outcome of one or more properties as functions of model and property parameters. The resulting table of values can either be viewed directly,

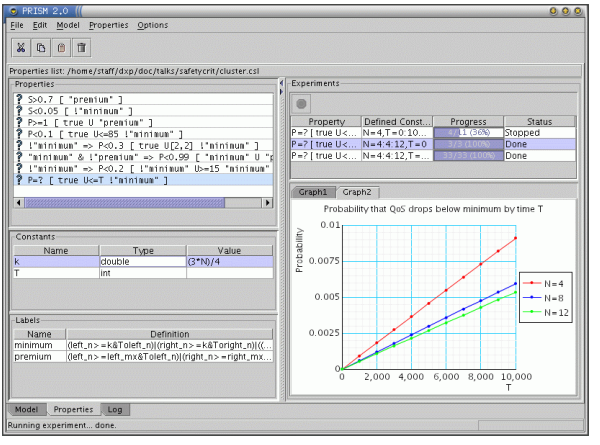


Fig. 5. A screenshot of the PRISM graphical user interface

exported for use in an external application such as a spreadsheet, or plotted as a graph. For the latter, PRISM incorporates substantial graph-plotting functionality. This is often a very useful way of identifying interesting patterns or trends in the behaviour of a system. The reader is invited to consult the 'Case Studies' section of the PRISM website [53] for many examples of this kind of analysis.

Fig. 5 shows a screenshot of the PRISM graphical user interface, illustrating the results of a model checking experiment being plotted on a graph. The tool also features a built-in text-editor for the PRISM language. Alternatively, all model checking functionality is also available in a command-line version of the tool. PRISM is a *free*, *open source* application. It presently operates on Linux, Unix, Windows and Macintosh operating systems. Both binary and source code versions can be downloaded from the website [53].

Implementation. One of the most notable features of PRISM is that it is a *symbolic* model checker, meaning that its implementation uses data structures based on binary decision diagrams (BDDs). These provide compact representations and efficient manipulation of large, structured probabilistic models by exploiting regularity that is often present in those models because they are described in a structured, high-level modelling language. More specifically, since we need to store numerical values, PRISM uses *multi-terminal* binary decision diagrams (MTBDDs) [21,7] and a number of variants [46,52,48] developed to improve the efficiency of probabilistic analysis, which involve combinations of *symbolic* data structures such as MTBDDs and conventional *explicit* storage schemes such as sparse matrices and arrays. Since its release in 2001, the model size capacity and tool efficiency has increased substantially (10^7 – 10^8 is feasible for CTMCs and higher for other types of models). PRISM employs and builds upon the Colorado University Decision Diagram package [58] by Fabio Somenzi which implements BDD/MTBDD operations.

The underlying computation in PRISM involves a combination of:

- *graph-theoretical algorithms*, for reachability analysis, conventional temporal logic model checking and *qualitative* probabilistic model checking;
- *numerical computation*, for *quantitative* probabilistic model checking, e.g. solution of linear equation systems (for DTMCs and CTMCs) and linear optimisation problems for (MDPs).

Graph-theoretical algorithms are comparable to the operation of a conventional, non-probabilistic model checker and are always performed in PRISM using BDDs. For numerical computation, PRISM uses iterative methods rather than direct methods due to the size of the models that need to be handled. For solution of linear equation systems, it supports a range of well-known techniques, including the Jacobi, Gauss-Seidel and SOR (successive over-relaxation) methods. For the linear optimisation problems which arise in the analysis of MDPs, PRISM uses dynamic programming techniques, in particular, value iteration. Finally, for transient analysis of CTMCs, PRISM incorporates another iterative numerical method, uniformisation (see Section 4.4).

In fact, for numerical computation, the tool actually provides three distinct numerical *engines*. The first is implemented purely in MTBDDs (and BDDs); the second uses sparse matrices; and the third is a hybrid, using a combination of the two. Performance (time and space) of the tool may vary depending on the choice of the engine. Typically the sparse engine is quicker than its MTBDD counterpart, but requires more memory. The hybrid engine aims to provide a compromise, providing faster computation than pure MTBDDs but using less memory than sparse matrices (see [46,52]).

The PRISM modelling language. The PRISM modelling language is a simple, state-based language based on the Reactive Modules formalism of Alur and Henzinger [1]. In this section, we give a brief outline of the language. For a full definition of the language and its semantics, see [45]. In addition a wide range of examples can be found both in the 'Case Studies' section of the PRISM website [53] and in the distribution of the tool itself.

The fundamental components of the PRISM language are *modules* and *variables*. Variables are typed (integers, reals and booleans are supported) and can be local or global. A model is composed of *modules* which can interact with each other. A module contains a number of local *variables*. The values of these variables at any given time constitute the state of the module. The *global state* of the whole model is determined by the *local state* of all modules, together with the values of the global variables. The behaviour of each module is described by a set of *commands*. A command takes the form:

$$[]\ g \rightarrow \lambda_1 : u_1 + \cdots + \lambda_n : u_n\,;$$

The *guard* g is a predicate over all the variables in the model (including those belonging to other modules). Each *update* u_i describes a transition which the module can make if the guard is true. A transition is specified by giving the new

```
dtmc

module D1

    x : [0..3] init 0;

    [] x=0 → (x'=1);
    [] x=1 → 0.01 : (x'=1)
           + 0.01 : (x'=2)
           + 0.98 : (x'=3);
    [] x=2 → (x'=0);
    [] x=3 → (x'=3);

endmodule
```

```
ctmc

module C1

    y : [0..3] init 0;

    [] y<3 → 1.5 : (y'=y+1);
    [serve] y>0 → 3 : (y'=y−1);

endmodule
```

Fig. 6. The PRISM Language: Specification of $\mathcal{D}_1$ and $\mathcal{C}_1$

values of the variables in the module, possibly as an *expression* formed from other variables or constants. The expressions λ_i are used to assign probabilistic information to the transitions.

In Fig. 6 we present the specification of the DTMC $\mathcal{D}_1$ (see Example 1 and Fig. 1) and the CTMC $\mathcal{C}_1$ (see Example 12 and Fig. 4). For both these models there is a single initial state, but PRISM allows the specification of a set of initial states, see [45]. The labelling '*serve*' of the second command in the specification of $\mathcal{C}_1$ will be used below to specify a reward structure for this model.

In general the probabilistic model corresponding to a PRISM language description is constructed as the parallel composition of its modules. In every state of the model, there is a set of commands (belonging to any of the modules) which are enabled, i.e. whose guards are satisfied in that state. The choice between which command is performed (i.e. the scheduling) depends on the model type. For a DTMC, the choice is *probabilistic*, with each enabled command selected with equal probability and for CTMCs it is modelled as a *race condition*. PRISM also supports multi-way *synchronisation* in the style of process algebras. For synchronisation to take effect, commands are labelled with *actions* that are placed between the square brackets.

Reward Structures. PRISM includes support for the specification and analysis of properties based on *reward* (and cost) structures. Reward structures are associated with models using the `rewards` "*reward_name*" ... `endrewards` construct and are specified using multiple reward items of the form:

$$g \ : \ r; \quad \text{or} \quad [a] \ g \ : \ r;$$

depending on whether a state or transition rewards are being specified, where g is a predicate (over all the variables of the model), a is a action label appearing in the commands of the model and r is a real-valued expression (containing any variables, constants, etc. from the model). A single reward item can assign

different rewards to different states or transitions, depending on the values of model variables in each one. Any states/transitions which do not satisfy the guard of a reward item will have no reward assigned to them. For states/transitions which satisfy multiple guards, the reward assigned is the sum of the rewards for all the corresponding reward items.

For example, the two reward structures of the CTMC $\mathcal{C}_1$ given in Example 18 can be specified as:

```
rewards "reward1"
    true : y;
endrewards
```

```
rewards "reward2"
    [serve] true : 1;
endrewards
```

To further illustrate how reward structures are specified in PRISM consider the reward structure given below, which assigns a state reward of 100 to states satisfying x=1 or y=1 and 200 to states that satisfy both x=1 and y=1, and a transition reward of $2 \cdot x$ to transitions labelled by a from states satisfying x>0 and x<5.

```
rewards "reward_name"
    x=1 : 100;
    y=1 : 100;
    [a] x>0 & x<5 : 2 * x;
endrewards
```

Property specifications. Properties of PRISM models are expressed in PCTL for DTMCs and CSL for CTMCs. The operators $\mathtt{P}_{\sim p}[\cdot]$, $\mathtt{S}_{\sim p}[\cdot]$ and $\mathtt{R}_{\sim r}[\cdot]$ by default include the probability bound $\sim p$ or reward bound $\sim r$. However, in PRISM, we can also directly specify properties which evaluate to a *numerical value* by replacing the bounds in the P, S and R operators with =?, as illustrated in the following PRISM specifications:

- P=? [! *proc2_terminate* U *proc1_terminate*] - the probability that process 1 terminates before process 2 completes;
- S=? [(*queue_size*/*max_size*)>0.75] - the long-run probability that the queue is more than 75% full;
- R=? [C ⩽ 24] - the expected power consumption during the first 24 hours of operation;
- R=? [I = 100] - after 100 time units, the expected number of packets awaiting delivery;
- R=? [F *elected*] - the expected number of steps required for the leader election algorithm to complete;
- R=? [S] - the long-run expected queue-size.

Note that the meaning ascribed to these properties is, of course, dependent on the definitions of the atomic propositions and reward structures.

By default, the result for properties of this kind is the probability for the initial state of the model. It is also possible, however, to obtain the probability

for an arbitrary state or more generally either the minimum or maximum probability for a particular class of states, as demonstrated in the following PRISM specifications:

- P=? [*queue_size*$\leqslant$5 U *queue_size*<5 {*queue_size*=5}] - the probability, from the state where the queue contains 5 jobs, of the queue processing at least one job before another arrives;
- P=? [!*proc2_terminate* U *proc1_terminate* {*init*}{min}] - the minimum probability, over all possible initial configurations, that process 1 terminates before process 2 does.

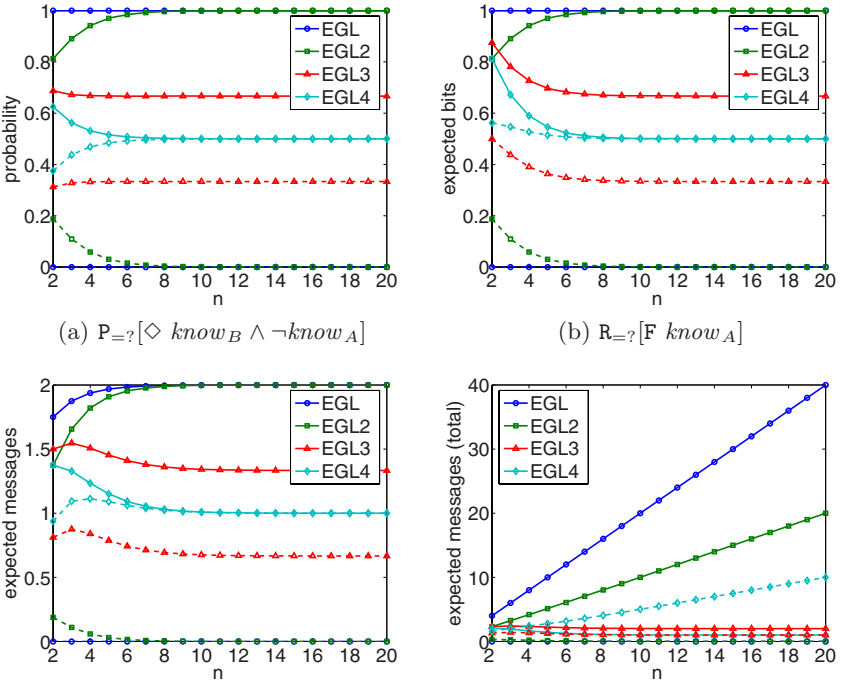
5.2 Case Study 1: Probabilistic Contract Signing

This case study, taken from [51], concerns the probabilistic contract signing protocol of Even, Goldreich and Lempel [25]. The protocol is designed to allow two parties, A and B, to exchange commitments to a contract. In an asynchronous setting, it is difficult to perform this task in a way that is fair to both parties, i.e. such that if B has obtained A's commitment, then A will always be able to obtain B's. In the Even, Goldreich and Lempel (EGL) protocol, the parties A and B each generate a set of pairs of secrets which are then revealed to the other party in a probabilistic fashion. A is committed to the contract once B knows both parts of one of A's pairs of secrets (and vice versa).

PRISM was used to identify a weakness of the protocol [51,53], showing that, by quitting the protocol early, one of the two parties (the one which did not initiate the protocol) can be at an advantage by being in possession of a complete pair of secrets while the other party knows no complete pairs. Various modifications to the basic EGL protocol were proposed [51,53] and PRISM was used to quantify the fairness of each.

The model is constructed as a DTMC and below we list the range of PCTL properties relating to party A that have been studied with PRISM (the dual properties for party B have also been studied). For each property we also state any modification to the model or reward structure required and explain the relevance of the property to the performance of the protocol.

- $\mathtt{P}_{=?}[\Diamond\ know_B \wedge \neg know_A]$ – the probability of reaching a state where A does not know a pair while B does know a pair. This measure can be interpreted as the "chance" that the protocol is unfair towards either party.
- $\mathtt{R}_{=?}[\mathtt{F}\ done]$ - the expected number of bits that A needs to know a pair once B knows a pair. In this case the model of the protocol was modified by adding a transition to the final state *done* as soon as B knows a pair and assigning to this transition a reward equal to the number of bits that A requires to know a pair. This property is a quantification of how unfair the protocol is with respect to either party.
- $\mathtt{R}_{=?}[\mathtt{F}\ know_A]$ - once B knows a pair, the expected number of messages from B that A needs to know a pair. The reward structure in this case associates a reward of 1 to all transitions which correspond to B sending a message to A



(a) $\mathtt{P}_{=?}[\Diamond\ know_B \wedge \neg know_A]$ (b) $\mathtt{R}_{=?}[\mathtt{F}\ know_A]$

(c) $\mathtt{R}_{=?}[\mathtt{F}\ know_A]$ (d) $\mathtt{R}_{=?}[\mathtt{F}\ know_A]$ (total messages)

Fig. 7. Model checking results for the EGL contract signing protocol

from a state where B already knows a pair. This measure can be interpreted as an indication of how much influence a corrupted party has on the fairness of the protocol, since a corrupted party can try and delay these messages in order to gain an advantage.

– $\mathtt{R}_{=?}[\mathtt{F}\ know_A]$ - once B knows a pair, the expected total number of messages that need to be sent (by either party) before A knows a pair. In this case we assign a reward of 1 to any transition which corresponds to either B sending a message to A or A sending a message to B in a state where B already knows a pair. This measure can be interpreted as representing the "duration" of unfairness, that is, the time that one of the parties has an advantage.

Fig. 7 shows plots of these values for both the basic protocol (EGL) and three modifications (EGL2, EGL3 and EGL4). The solid lines and dashed lines represent the values for parties A and B, respectively (where process B initiated the protocol). The data is computed for a range of values of n: the number of pairs of secrets which each party generates.

The results show EGL4 is the 'fairest' protocol except for the 'duration of fairness measure' (expected messages that need to be sent for a party to know a

pair once the other party knows a pair). For this measure, the value is larger for B than for A and, in fact, as n increases, this measure increases for B but decreases for A. In [51] a solution is proposed and analysed which merges sequences of bits into a single message. For further details on this case study see [51] and the PRISM website [53].

5.3 Case Study 2: Dynamic Power Management

Dynamic Power Management(DPM) is a technique for saving energy in devices which can be turned on and off under operating system control. DPM has gained considerable attention over the last few years, both in the research literature and in the industrial setting, with schemes such as OnNow and ACPI becoming prevalent. One of the main reasons for this interest is the continuing growth in the use of mobile, hand-held and embedded devices, for which minimisation of power consumption is a key issue.

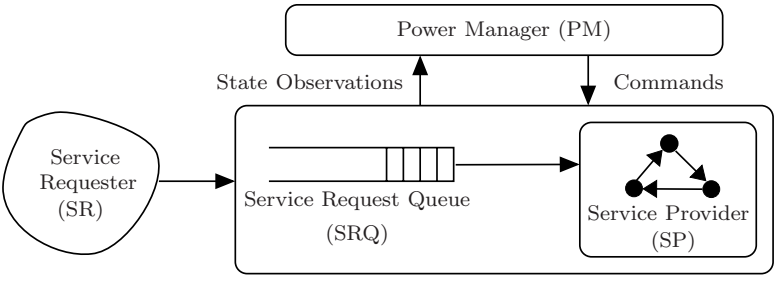


Fig. 8. The DPM System Model

DPM-enabled devices typically have several *power states* with different power consumption rates. A DPM *policy* is used to decide when commands to transition between these states should be issued, based on the current state of the system. In this case study we consider only simple policies, so called N-policies, which 'switch on' when the queue of requests awaiting service is greater than or equal to N, and 'switch off' when the queue becomes empty.

The basic structure of the DPM model can be seen in Fig. 8. The model consists of: a Service Provider (SP), which represents the device under power management control; a Service Requester (SR), which issues requests to the device; a Service Request Queue (SRQ), which stores requests that are not serviced immediately; and the Power Manager (PM), which issues commands to the SP, based on observations of the system and a stochastic DPM policy.

This case study is based on a CTMC model of a Fujitsu disk drive [54]. The SP has three power states: *sleep*, *idle* and *busy*. In *sleep* the SP is inactive and no requests can be served. In *idle* and *busy* the SP is active; the difference is that *idle* corresponds to the case when the SP is not working on any requests (the SRQ is empty) and *busy* it is actively working on requests (the SRQ is not empty). Transitions between *sleep* and *idle* are controlled by the PM (that is,

	sleep	*idle*	*busy*
sleep	0	1.6	-
idle	0.67	0	0
busy	-	0	0

(a) Transition time

	sleep	*idle*	*busy*
sleep	0	7	-
idle	0.067	0	0
busy	-	0	0

(b) Energy consumed

	sleep	*idle*	*busy*
av. power	0.13	0.95	2.15
av. service	0	0	0.008

(c) Power and service times

Fig. 9. Transition times, energy and power consumption and service times for the SP

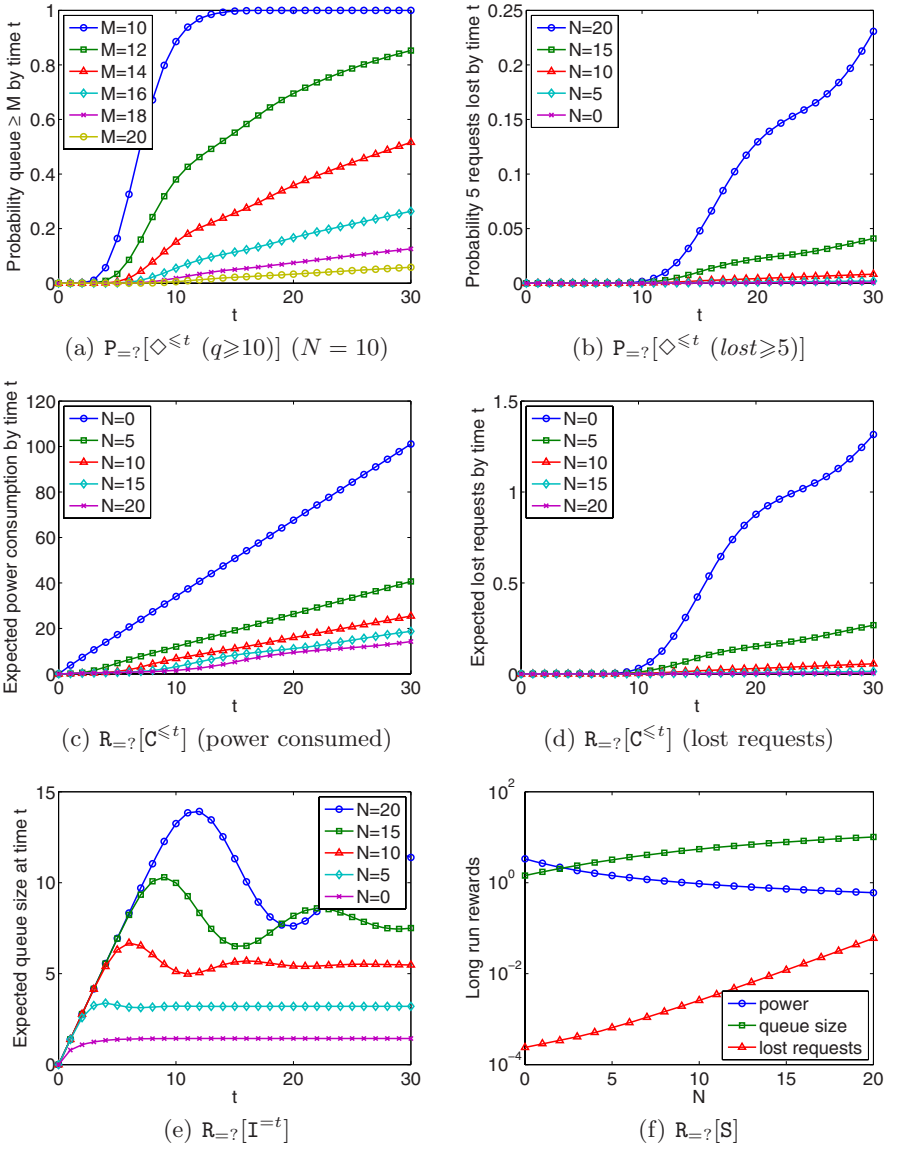
by the DPM policy), while transitions between *idle* and *busy* are controlled by the state of the SRQ. Fig. 9(a) shows the average times for transitions between power states, Fig. 9(b) show the energy used for these transitions and Fig. 9(c) the average power consumption and service times for each state. The SR models the inter-arrival distribution of requests given by exponential distribution with rate 100/72 and the SRQ models a service request queue which has a maximum size of 20. Note that, if a request arrives from the SR and the queue is full (20 requests are already awaiting service), then they are presumed lost.

The three reward structures constructed for this case study which are outlined below.

1. The first reward structure, used to investigate the power consumption of the system, is defined using the energy and power consumption of the SP given in Fig. 9. More precisely, the state rewards equal the average power consumption of the SP in that state and the transition reward for transitions in which the SP changes state is assigned the energy consumed by the corresponding state change.
2. The second reward structure, used for analysing the size of the service request queue, is obtained by setting the reward in each state to the size of the SRQ in that state (there are no transition based rewards);
3. The third reward structure, used when calculating the number of lost requests, assigns a reward of 1 to any transition representing the arrival of a request in a state where the queue is full (there are no state rewards in this case).

Below we list a range of CSL properties that have been studied for this case study in PRISM.

- $\mathtt{P}_{=?}[\Diamond^{\leqslant t}\ (q \geqslant M)]$ – the probability that the queue size becomes greater than or equal to M by t;
- $\mathtt{P}_{=?}[\Diamond^{\leqslant t}\ (\mathit{lost} \geqslant M)]$ – the probability that at least M requests get lost by t;
- $\mathtt{R}_{=?}[\mathtt{C}^{\leqslant t}]$ – the expected power consumption by t or the expected number of lost customers by time t (depending on whether the first or third reward structure is used);
- $\mathtt{R}_{=?}[\mathtt{I}^{=t}]$ – the expected queue size at t (using the second reward structure);
- $\mathtt{R}_{=?}[\mathtt{S}]$ – the long run average power consumption, long run average queue size or long run average number of requests lost per unit time (depending on which reward structure is used).



(a) $\mathtt{P}_{=?}[\Diamond^{\leqslant t}\ (q{\geqslant}10)]$ ($N = 10$)

(b) $\mathtt{P}_{=?}[\Diamond^{\leqslant t}\ (\mathit{lost}{\geqslant}5)]$

(c) $\mathtt{R}_{=?}[\mathtt{C}^{\leqslant t}]$ (power consumed)

(d) $\mathtt{R}_{=?}[\mathtt{C}^{\leqslant t}]$ (lost requests)

(e) $\mathtt{R}_{=?}[\mathtt{I}^{=t}]$

(f) $\mathtt{R}_{=?}[\mathtt{S}]$

Fig. 10. Range of results for the DPM case study obtained with PRISM

Fig. 10 presents a range of the results obtained with PRISM for this case study. The results demonstrate, as expected, that increasing N decreases the power consumption, while increasing both the queue size and the number of lost requests. For further details about DPM see, for example, [14,55] and for probabilistic model checking of DPM [50].

5.4 Case Study 3: Fibroblast Growth Factors

The final case study concerns a biological pathway for Fibroblast Growth Factors taken from [32]. Fibroblast Growth Factors (FGF) are a family of proteins which play a key role in the process of cell signalling in a variety of contexts, for example wound healing. The model is a CTMC and it incorporates protein-protein interactions (including competition for partners), phosphorylation and dephosphorylation, protein complex relocation and protein complex degradation (via ubiquitin-mediated proteolysis). Fig. 11 illustrates the different components in the pathway and their possible bindings.

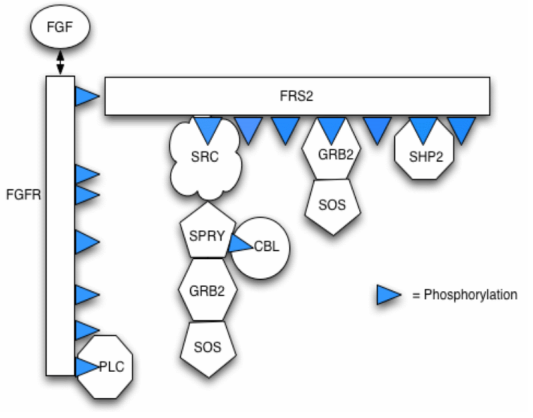
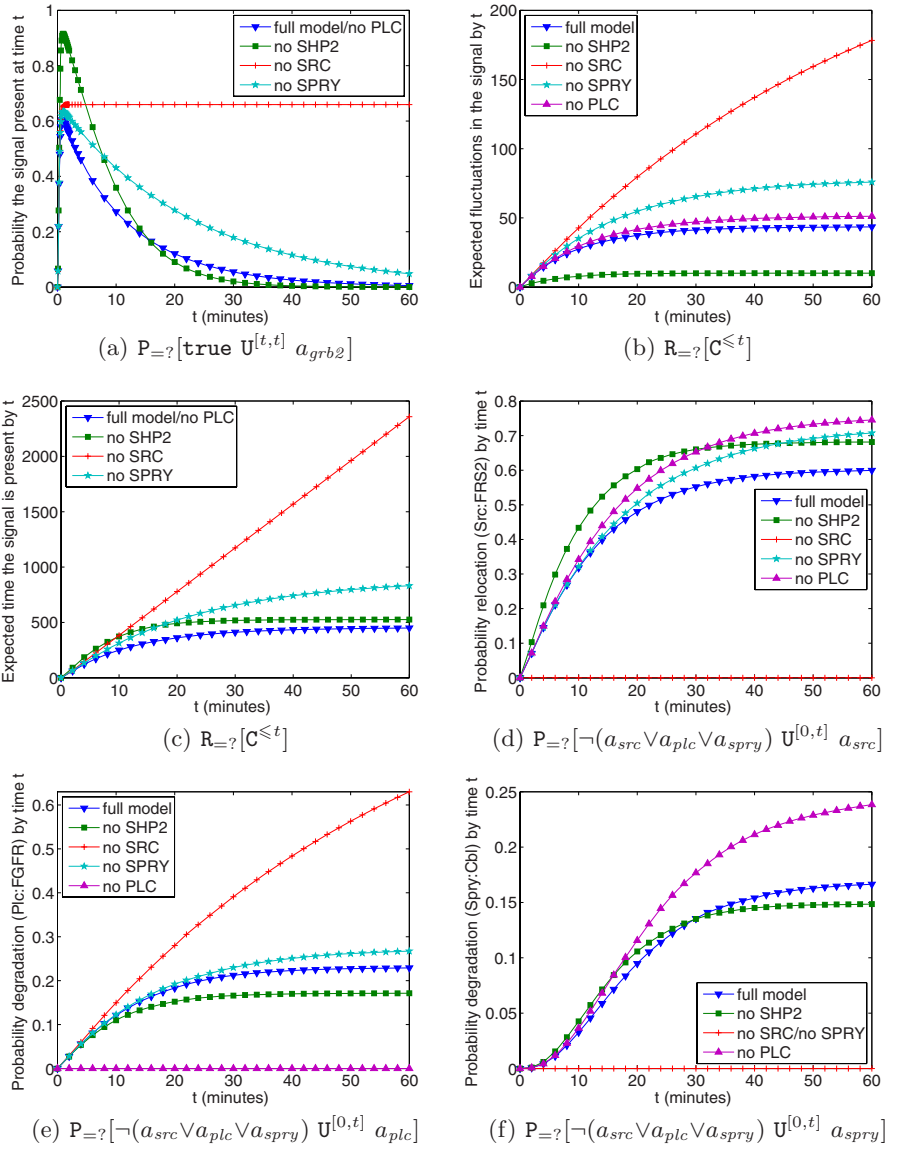


Fig. 11. Diagram showing the different possible bindings in the pathway

In [32] a base model, representing the full system, was developed. Subsequently, a series of 'in silico genetics' experiments on the model designed to investigate the roles of the various components of the activated receptor complex in controlling signalling dynamics. This involves deriving a series of modified models of the pathway where certain components are omitted (Shp2, Src, Spry or Plc), and is easily achieved in a PRISM model by just changing the initial value of the component under study. Below, we present a selection of the various CSL properties of the model that were analysed including, for properties relating to rewards, an explanation of the corresponding reward structure.

- $\mathtt{P}_{=?}[\Diamond^{[t,t]}\ a_{grb2}]$ - the probability that Grb2 is bound to FRS2 at the time instant t.
- $\mathtt{R}_{=?}[\mathtt{C}^{\leqslant t}]$ - the expected number of times that Grb2 binds to FRS2 by time t. In this case, the only non-zero rewards are associated with transitions involving Grb2 binding to FRS2 which have a reward 1.
- $\mathtt{R}_{=?}[\mathtt{C}^{\leqslant t}]$ – the expected time that Grb2 spends bound to FRS2 within the first T time units. The reward structure for this property assigns a reward of 1 to all states where Grb2 is bound to FRS2 and 0 to all other states and transitions.
- $\mathtt{S}_{=?}[a_{grb2}]$ - the long-run probability that Grb2 is bound to FRS2.



(a) $\mathtt{P}_{=?}[\mathtt{true}\ \mathtt{U}^{[t,t]}\ a_{grb2}]$ (b) $\mathtt{R}_{=?}[\mathtt{C}^{\leqslant t}]$

(c) $\mathtt{R}_{=?}[\mathtt{C}^{\leqslant t}]$ (d) $\mathtt{P}_{=?}[\neg(a_{src}\vee a_{plc}\vee a_{spry})\ \mathtt{U}^{[0,t]}\ a_{src}]$

(e) $\mathtt{P}_{=?}[\neg(a_{src}\vee a_{plc}\vee a_{spry})\ \mathtt{U}^{[0,t]}\ a_{plc}]$ (f) $\mathtt{P}_{=?}[\neg(a_{src}\vee a_{plc}\vee a_{spry})\ \mathtt{U}^{[0,t]}\ a_{spry}]$

Fig. 12. Transient numerical results

- $\mathtt{R}_{=?}[\mathtt{F}\ (a_{src}\vee a_{plc}\vee a_{spry})]$ - the expected number of times Grb2 binds to FRS2 before degradation or relocation occurs. As in the second property, transitions involving Grb2 binding to FRS2 are assigned reward 1.
- $\mathtt{R}_{=?}[\mathtt{F}\ (a_{src}\vee a_{plc}\vee a_{spry})]$ - the expected time Grb2 spends bound to FRS2 before degradation or relocation occurs As for the third property, all states where Grb2 is bound to FRS2 have a reward of 1.

Table 1. Long run and expected reachability properties for the signal

	$\mathtt{S}_{=?}[a_{grb2}]$	$\mathtt{R}_{=?}[\mathtt{F}\ (a_{src} \vee a_{plc} \vee a_{spry})]$	
		bindings	time (min)
full model	7.54e-7	43.1027	6.27042
no Shp2	3.29e-9	10.0510	7.78927
no Src	0.659460	283.233	39.6102
no Spry	4.6e-6	78.3314	10.8791
no Plc	0.0	51.5475	7.56241

Table 2. Probability and expected time until degradation/relocation in the long run

	$\mathtt{P}_{=?}[\neg(a_{src} \vee a_{plc} \vee a_{spry})\ \mathtt{U}\ a_{xxx}]$			$\mathtt{R}_{=?}[\mathtt{F}\ (a_{src} \vee a_{plc} \vee a_{spry})]$
	$xxx = src$	$xxx = plc$	$xxx = spry$	(min)
full model	0.602356	0.229107	0.168536	14.0258
no Shp2	0.679102	0.176693	0.149742	10.5418
no Src	-	1.0	0.0	60.3719
no Spry	0.724590	0.275410	-	16.8096
no Plc	0.756113	-	0.243887	17.5277

- $\mathtt{P}_{=?}[\neg(a_{src} \vee a_{plc} \vee a_{spry})\ \mathtt{U}^{[0,t]}\ a_{src}]$ - the probability that degradation or relocation occurs by time t and Src is the cause.
- $\mathtt{P}_{=?}[\neg(a_{src} \vee a_{plc} \vee a_{spry})\ \mathtt{U}\ a_{plc}]$ - the probability that Plc is the first cause of degradation or relocation.
- $\mathtt{R}_{=?}[\mathtt{F}\ (a_{src} \vee a_{plc} \vee a_{spry})]$ - the expected time until degradation or relocation occurs in the pathway. For this property all states are assigned reward 1 (and all transitions are assigned reward 0).

Fig. 12 presents results relating to the transient properties, while Tables 1 and 2 consider long-run properties. Note that the results of Table 1 and Table 2 can be regarded as the values of Fig. 12(a)–(c) and Fig. 12(d)–(f) in "the limit", i.e. as t tends to infinity. For further details on the case study see [32] and the PRISM website [53].

6 Conclusions

In this tutorial we have presented an overview of stochastic model checking, covering both the theory and practical aspects for two important types of probabilistic models, discrete- and continuous-time Markov chains. Algorithms were given for verifying these models against probabilistic temporal logics PCTL and CSL and their extensions with the reward operator. The probabilistic model checker PRISM, which implements these algorithms, was used to analyse three real-world case studies also described here. However, there are many other aspects of stochastic model checking not covered in this tutorial and below we attempt to give brief pointers to related and further work.

More expressive logics than PCTL have been proposed, including LTL and PCTL* [6,15]. For the corresponding model checking algorithms see [62,22,6,15,13]. We also mention the alternative reward extension of PCTL given in [2]. With regards to CTMCs, a number of extensions of CSL have been proposed in the literature, along with associated model checking algorithms. For example, [35] proposes an action based version of CSL; [31,9] introduce the logics CRL and CSRL which added support for reward-based properties [42]; and [44] augment CSL with random time-bounded until and random expected-time operators, respectively.

This tutorial concentrated on stochastic model checking. Related topic include: probabilistic generalisations of bisimulation and simulation relations for DTMCs [49,57] and for CTMCs [17,11]; and approximate methods for stochastic model checking based on discrete event simulation [33,63]. Stochastic model checkers SMART [19], E⊢MC^2 [34] and MRMC [39] have similarities with the PRISM model checker described here. Finally, we mention a challenging direction of research is into the verification of models which allow more general probability distributions. While the restriction to exponential distributions imposed by CTMCs is important for the tractability of their model checking, it may prove too simplistic for some modelling applications. See [28] for an introduction to this area.

References

1. R. Alur and T. Henzinger. Reactive modules. *Formal Methods in System Design*, 15(1):7–48, 1999.
2. S. Andova, H. Hermanns, and J.-P. Katoen. Discrete-time rewards model-checked. In K. Larsen and P. Niebert, editors, *Proc. Formal Methods for Timed Systems (FORMATS'03)*, volume 2791 of *LNCS*, pages 88–104. Springer, 2003.
3. J. Aspnes and M. Herlihy. Fast randomized consensus using shared memory. *Journal of Algorithms*, 15(1):441–460, 1990.
4. A. Aziz, K. Sanwal, V. Singhal, and R. Brayton. Verifying continuous time Markov chains. In R. Alur and T. Henzinger, editors, *Proc. 8th Int. Conf. Computer Aided Verification (CAV'96)*, volume 1102 of *LNCS*, pages 269–276. Springer, 1996.
5. A. Aziz, K. Sanwal, V. Singhal, and R. Brayton. Model checking continuous time Markov chains. *ACM Transactions on Computational Logic*, 1(1):162–170, 2000.
6. A. Aziz, V. Singhal, F. Balarin, R. Brayton, and A. Sangiovanni-Vincentelli. It usually works: The temporal logic of stochastic systems. In P. Wolper, editor, *Proc. 7th Int. Conf. Computer Aided Verification (CAV'95)*, volume 939 of *LNCS*, pages 155–165. Springer, 1995.
7. I. Bahar, E. Frohm, C. Gaona, G. Hachtel, E. Macii, A. Pardo, and F. Somenzi. Algebraic decision diagrams and their applications. *Formal Methods in System Design*, 10(2/3):171–206, 1997.
8. C. Baier, B. Haverkort, H. Hermanns, and J.-P. Katoen. Model checking continuous-time Markov chains by transient analysis. In A. Emerson and A. Sistla, editors, *Proc. 12th Int. Conf. Computer Aided Verification (CAV'00)*, volume 1855 of *LNCS*, pages 358–372. Springer, 2000.

9. C. Baier, B. Haverkort, H. Hermanns, and J.-P. Katoen. On the logical characterisation of performability properties. In U. Montanari, J. Rolim, and E. Welzl, editors, *Proc. 27th Int. Colloquium on Automata, Languages and Programming (ICALP'00)*, volume 1853 of *LNCS*, pages 780–792. Springer, 2000.
10. C. Baier, B. Haverkort, H. Hermanns, and J.-P. Katoen. Model-checking algorithms for continuous-time Markov chains. *IEEE Transactions on Software Engineering*, 29(6):524–541, 2003.
11. C. Baier, J. Katoen, H. Hermanns, and B. Haverkort. Simulation for continuous-time Markov chains. In L. Brim, P. Jancar, M. Kretinzki, and A. Kucera, editors, *Proc. Concurrency Theory (CONCUR'02)*, volume 2421 of *LNCS*, pages 338–354. Springer, 2002.
12. C. Baier, J.-P. Katoen, and H. Hermanns. Approximate symbolic model checking of continuous-time Markov chains. In J. Baeten and S. Mauw, editors, *Proc. 10th Int. Conf. Concurrency Theory (CONCUR'99)*, volume 1664 of *LNCS*, pages 146–161. Springer, 1999.
13. C. Baier and M. Kwiatkowska. Model checking for a probabilistic branching time logic with fairness. *Distributed Computing*, 11(3):125–155, 1998.
14. L. Benini, A. Bogliolo, G. Paleologo, and G. D. Micheli. Policy optimization for dynamic power management. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 8(3):299–316, 2000.
15. A. Bianco and L. de Alfaro. Model checking of probabilistic and nondeterministic systems. In P. Thiagarajan, editor, *Proc. 15th Conf. Foundations of Software Technology and Theoretical Computer Science*, volume 1026 of *LNCS*, pages 499–513. Springer, 1995.
16. P. Billingsley. *Probability and Measure*. Wiley, 1995.
17. P. Buchholz. Exact and ordinary lumpability in finite Markov chains. *Journal of Applied Probability*, 31:59–75, 1994.
18. L. Cheung. Randomized wait-free consensus using an atomicity assumption. In *Proc. 9th International Conference on Principles of Distributed Systems (OPODIS'05)*, 2005.
19. G. Ciardo, R. Jones, A. Miner, and R. Siminiceanu. Logic and stochastic modeling with smart. *Performance Evaluation*, 63(6):578–608, 2006.
20. E. Clarke, E. Emerson, and A. Sistla. Automatic verification of finite-state concurrent systems using temporal logics. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, 1986.
21. E. Clarke, M. Fujita, P. McGeer, K. McMillan, J. Yang, and X. Zhao. Multi-terminal binary decision diagrams: An efficient data structure for matrix representation. *Formal Methods in System Design*, 10((2/3):149–169, 1997.
22. C. Courcoubetis and M. Yannakakis. Verifying temporal properties of finite state probabilistic programs. In *Proc. 29th Annual Symposium on Foundations of Computer Science (FOCS'88)*, pages 338–345. IEEE Computer Society Press, 1988.
23. C. Courcoubetis and M. Yannakakis. The complexity of probabilistic verification. *Journal of the ACM*, 42(4):857–907, 1995.
24. C. Daws, M. Kwiatkowska, and G. Norman. Automatic verification of the IEEE 1394 root contention protocol with KRONOS and PRISM. *Int. Journal on Software Tools for Technology Transfer*, 5(2–3):221–236, 2004.
25. S. Even, O. Goldreich, and A. Lempel. A randomized protocol for signing contracts. *Communications of the ACM*, 28(6):637–647, 1985.
26. W. Fokkink and J. Pang. Variations on itai-rodeh leader election for anonymous rings and their analysis in prism. *Journal of Universal Computer Science*, 12(8):981–1006, 2006.

27. B. Fox and P. Glynn. Computing Poisson probabilities. *Communications of the ACM*, 31(4):440–445, 1988.
28. R. German. *Performance Analysis of Communication Systems: Modeling with Non-Markovian Stochastic Petri Nets*. John Wiley and Sons, 2000.
29. H. Hansson and B. Jonsson. A logic for reasoning about time and reliability. *Formal Aspects of Computing*, 6(5):512–535, 1994.
30. B. Haverkort. *Performance of Computer Communication Systems: A Model-Based Approach*. John Wiley & Sons, 1988.
31. B. Haverkort, L. Cloth, H. Hermanns, J.-P. Katoen, and C. Baier. Model checking performability properties. In *Proc. Int. Conf. Dependable Systems and Networks (DSN'02)*. IEEE Computer Society Press, 2002.
32. J. Heath, M. Kwiatkowska, G. Norman, D. Parker, and O. Tymchyshyn. Probabilistic model checking of complex biological pathways. In C. Priami, editor, *Proc. Computational Methods in Systems Biology (CMSB'06)*, volume 4210 of *Lecture Notes in Bioinformatics*, pages 32–47. Springer, 2006.
33. T. Hérault, R. Lassaigne, F. Magniette, and S. Peyronnet. Approximate probabilistic model checking. In B. Steffen and G. Levi, editors, *Proc. Verification, Model Checking and Abstract Interpretation (VMCAI'04)*, volume 2937 of *LNCS*, pages 73–84. Springer, 2004.
34. H. Hermanns, J.-P. Katoen, J. Meyer-Kayser, and M. Siegle. A Markov chain model checker. In S. Graf and M. Schwartzbach, editors, *Proc. 6th Int. Conf. Tools and Algorithms for the Construction and Analysis of Systems (TACAS'00)*, volume 1785 of *LNCS*, pages 347–362. Springer, 2000.
35. H. Hermanns, J.-P. Katoen, J. Meyer-Kayser, and M. Siegle. Towards model checking stochastic process algebra. In W. Grieskamp and T. Santen, editors, *Proc. Integrated Formal Method (IFM 2000)*, volume 1945 of *LNCS*, pages 420–439. Springer, 2000.
36. A. Hinton, M. Kwiatkowska, G. Norman, and D. Parker. PRISM: A tool for automatic verification of probabilistic systems. In H. Hermanns and J. Palsberg, editors, *Proc. 12th Int. Conf. Tools and Algorithms for the Construction and Analysis of Systems (TACAS'06)*, volume 3920 of *LNCS*, pages 441–444. Springer, 2006.
37. IEEE standard for a high performance serial bus. IEEE Computer Society, IEEE Std 1394-1995.
38. A. Itai and M. Rodeh. Symmetry breaking in distributed networks. *Information and Computation*, 88(1):60–87, 1990.
39. J.-P. Katoen, M. Khattri, and I. Zapreev. A Markov reward model checker. In *Proc. Second Int. Conf. Quantitative Evaluation of Systems (QEST 05)*, pages 243–244. IEEE Computer Society Press, 2005.
40. J.-P. Katoen, M. Kwiatkowska, G. Norman, and D. Parker. Faster and symbolic CTMC model checking. In L. de Alfaro and S. Gilmore, editors, *Proc. 1st Joint Int. Workshop on Process Algebra and Probabilistic Methods, Performance Modeling and Verification (PAPM/PROBMIV'01)*, volume 2165 of *LNCS*, pages 23–38. Springer, 2001.
41. J. Kemeny, J. Snell, and A. Knapp. *Denumerable Markov Chains*. Springer, 2nd edition, 1976.
42. M. Kwiatkowska, G. Norman, and A. Pacheco. Model checking CSL until formulae with random time bounds. In H. Hermanns and R. Segala, editors, *Proc. 2nd Joint Int. Workshop on Process Algebra and Probabilistic Methods, Performance Modeling and Verification (PAPM/PROBMIV'02)*, volume 2399 of *LNCS*, pages 152–168. Springer, 2002.

43. M. Kwiatkowska, G. Norman, and A. Pacheco. Model checking expected time and expected reward formulae with random time bounds. In *Proc. 2nd Euro-Japanese Workshop on Stochastic Risk Modelling for Finance, Insurance, Production and Reliability*, 2002.
44. M. Kwiatkowska, G. Norman, and A. Pacheco. Model checking expected time and expected reward formulae with random time bounds. *Computers & Mathematics with Applications*, 51(2):305–316, 2006.
45. M. Kwiatkowska, G. Norman, and D. Parker. PRISM users' guide. Available from www.cs.bham.ac.uk/~dxp/prism.
46. M. Kwiatkowska, G. Norman, and D. Parker. Probabilistic symbolic model checking with PRISM: A hybrid approach. *Int. Journal on Software Tools for Technology Transfer*, 6(2):128–142, 2004.
47. M. Kwiatkowska, G. Norman, D. Parker, and J. Sproston. Performance analysis of probabilistic timed automata using digital clocks. *Formal Methods in System Design*, 29:33–78, 2006.
48. M. Kwiatkowska, D. Parker, Y. Zhang, and R. Mehmood. Dual-processor parallelisation of symbolic probabilistic model checking. In D. DeGroot and P. Harrison, editors, *Proc. 12th Int. Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS'04)*, pages 123–130. IEEE Computer Society Press, 2004.
49. K. Larsen and A. Skou. Bisimulation through probabilistic testing. *Information and Computation*, 94:1–28, 1991.
50. G. Norman, D. Parker, M. Kwiatkowska, S. Shukla, and R. Gupta. Using probabilistic model checking for dynamic power management. *Formal Aspects of Computing*, 17(2):160–176, 2005.
51. G. Norman and V. Shmatikov. Analysis of probabilistic contract signing. *Journal of Computer Security*, 14(6):561–589, 2006.
52. D. Parker. *Implementation of Symbolic Model Checking for Probabilistic Systems.* PhD thesis, University of Birmingham, 2002.
53. PRISM web site. www.cs.bham.ac.uk/~dxp/prism.
54. Q. Qiu, Q. Wu, and M. Pedram. Stochastic modeling of a power-managed system: Construction and optimization. In *Proc. Int. Symposium on Low Power Electronics and Design*, 1999.
55. Q. Qiu, Q. Wu, and M. Pedram. Stochastic modeling of a power-managed system: construction and optimization. *IEEE Transactions on Computer Aided Design*, 20(10):1200–1217, 2001.
56. J. Rutten, M. Kwiatkowska, G. Norman, and D. Parker. *Mathematical Techniques for Analyzing Concurrent and Probabilistic Systems,* P. Panangaden and F. van Breugel (eds.), volume 23 of *CRM Monograph Series.* American Mathematical Society, 2004.
57. R. Segala and N. Lynch. Probabilistic simulations for probabilistic processes. In B. Jonsson and J. Parrow, editors, *Proc. 5th Int. Conf. Concurrency Theory (CONCUR'94)*, volume 836 of *LNCS*, pages 481–496. Springer, 1994.
58. F. Somenzi. CUDD: Colorado University decision diagram package. Public software, Colorado Univeristy, Boulder, http://vlsi.colorado.edu/~fabio/, 1997.
59. W. J. Stewart. *Introduction to the Numerical Solution of Markov Chains.* Princeton, 1994.
60. R. Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1:146–160, 1972.
61. K. Trivedi. *Probability and Statistics with Reliability, Queuing, and Computer Science Applications.* John Wiley & Sons, 2001.

62. M. Vardi. Automatic verification of probabilistic concurrent finite state programs. In *Proc. 26th Annual Symposium on Foundations of Computer Science (FOCS'85)*, pages 327–338. IEEE Computer Society Press, 1985.
63. H. Younes, M. Kwiatkowska, G. Norman, and D. Parker. Numerical vs. statistical probabilistic model checking. *Int. Journal on Software Tools for Technology Transfer*, 8(3):216–228, 2006.

Fluid Models in Performance Analysis*

Marco Gribaudo[1] and Miklós Telek[2]

[1] Dip. di Informatica, Università di Torino
marcog@di.unito.it
[2] Dept. of Telecom., Technical University of Budapest
telek@hitbme.hu

Abstract. Stochastic fluid models have been applied to model and evaluate the performance of many important real systems. The automatic analysis tools to support of fluid models are still not as improved as the ones for discrete state Markov models, but there is a wide range of models which can be effectively described and analyzed with fluid models. Also the model support of hybrid models from various performance evaluation tools improves continuously.

The aim of this work is to summarize the basic concepts and the potential use of Markov fluid models. The factors which determine the limits of solvability of fluid models are also discussed. Practical guidelines can be extracted from these factors to determine the applicability of fluid models in practical modeling examples. The work is supported by an example where Fluid Models, derived from an higher level modeling language (Fluid Stochastic Petri Nets), have been exploited to study the transfer time distribution in Peer-to-Peer file sharing applications.

1 Introduction

Fluid models describe systems using two different kinds of variables; the discrete variables and the continuous variables. Usually the state of a fluid model can be decomposed into a discrete part which takes into account only the values of the discrete variables, and a fluid part, which considers only the changes in the fluid variables. The aim of this work is to summarize the basic concepts and the potential use of Markov fluid models, and to discuss the factors which determine the limits of solvability of this kind models.

This work is structured as follows: in Section 2, the main motivations and advantages of fluid models are presented. The main literature about fluid models is analyzed in 3. Section 4 defines three different formalisms that can be used to express fluid models. Section 5 deals with the analytical description of fluid models for steady state and transient analysis. Solution techniques are addressed in Section 6. In Section 7 an application of Fluid Stochastic Petri Nets to compute the transfer time distribution in Peer-to-Peer file sharing systems is presented. Section 8 concludes the work.

* This work is partially supported by the Italian-Hungarian R&D project 9/2003 and by the OTKA K61709 grant.

M. Bernardo and J. Hillston (Eds.): SFM 2007, LNCS 4486, pp. 271–317, 2007.

2 Motivations

Even if the conventional performance evaluation techniques are well suited to describe a wide range of real systems, things are not always as easy as they seem. The modeler usually faces several problems when trying to describe a system with a particular formalism, and sometimes these problems make the models extremely hard to handle. Some examples are:

- **State space explosion.** One of the weakest points in performance evaluation is that the complexity of the solution of discrete state models generally grows exponentially with the complexity of the model behavior. Many analysis techniques for most of the formalisms require the generation and the visit of all the possible states that the system may reach. This set of states is called the *state space of the model*, and for many applications it must be stored in the central memory. Since it grows exponentially with the complexity of the model, the size of this set may reach very quickly the storage capacity of the machine. In many situations this problem prevents a well defined model from being solved.
- **Inaccurate results.** A model is always a simplification of realty. Some simplifications are motivated by the fact that they cut out some parameters of the model that do not influence the required solutions. Some others are required in order to produce a system that can be analyzed with the tools that a modeler has. These simplifications may not be adequate sometimes and can lead to incorrect or inaccurate results. Many of these simplifications involve the characterization of some stochastic process by a Poisson process and the probability distribution of the time between two events by an exponential distribution.
- **Granularity and sizes.** In many situations the user must deal with a huge number of small elements. Let us consider for example a production line that produces bolts and screws. Thousands of parts will be produced in a very short time. A model that wishes to capture the number of parts produced, must deal with this big number which usually makes the state space explode even faster. Similar problems arise in today's communication systems which deals with a high number of very small data packets.
- **Modeling power limitations.** Sometimes a model depends on some physical quantity such as temperature or power consumption. Those are continuous quantities and they cannot be emulated correctly by discrete states. In these cases, the modeling power of a discrete state model specification language may not be adequate to describe the system.

2.1 Possible Solutions

In order to overcome the mentioned limitations, new modeling techniques have been developed. In this paper, we will examine how the previous problems can be attacked using *Hybrid continuous / discrete techniques*. Continuous and hybrid models can in some circumstances solve the above mentioned problems, or give

better results than conventional discrete state techniques in terms of computational complexity or accuracy of results. For example, hybrid models may solve the problems in the following way:

- *State space explosion*: Usually hybrid models are analyzed by splitting the discrete state space into a *discrete part*, that takes into account the possible states that the system may reach (by considering only the discrete components of the system) and a *continuous part*. Usually, when solving a hybrid model, only the discrete part of the state space must be memorized explicitly, while the continuous part is expressed as a set of functions or predicates. This greatly reduces the number of states that must be memorized and in some cases may solve the state space explosion problem.
- *Inaccurate results*: When the inaccuracy of a result is caused by the Poisson arrival or the exponential service time distribution, continuous models can be used to overcome this problem. Continuous components can be used to explicitly model the time since an action has been enabled, and can thus be used to model non-Markovian processes and complex memory properties.
- *Granularity and sizes*: When the modeling problems are caused by a variable that has a very small granularity, this variable may be approximated by a continuous quantity. Even if in the real system the variable is actually discrete, usually its continuous approximation can lead to very good results, especially if the changes in the real quantity happens at a constant rate. For example modeling the number of packets contained in a queue of an ATM router or the number of bytes allocated in the central memory of a PC can produce good results .
- *Modeling power limitations*: If the system under study depends on a physical continuous quantity that must be modeled explicitly to capture its real behavior, then hybrid models seem to be the natural solution. For example the instantaneous fuel consumption of a turbine in a power plant can be modeled explicitly by a continuous variable.

3 Related Works

Several formalisms have been introduced in the literature with this purpose in view. We will start our work with a review of the ones related to the results presented in the following sections.

Fluid Stochastic Petri Nets. Fluid Stochastic Petri Nets were introduced by K. S. Trivedi and V. G. Kulkarni in [37], and the extended in [21]. An FSPN is an extension of stochastic Petri net in which continuous quantities may be included directly in the model. A FSPN has two type of places: discrete places (containing a non-negative number of tokens) and continuous places (containing fluid). Discrete places are drawn as single circle, while fluid places are represented by two concentric circles. Transition firings are determined by both discrete and continuous places, and fluid flow is permitted through enabled timed transitions. Transitions may be either timed, when they have associated an exponential firing

time, or immediate, when they fire in zero time. To each transition a *guard* may be associated. A guard is boolean function of the state space $\mathcal{G} : (\mathcal{T} \times \mathcal{S}) \rightarrow \{0, 1\}$ where $\mathcal{T}$ represents the set of transitions and $\mathcal{S}$ the discrete-continuous state space. A transition t may fire in state $\mathcal{M}$ only if $\mathcal{G}(t, \mathcal{M}) = 1$. Guards associated with timed transitions may depend upon the marking of both the discrete and the continuous places, while guards associated with immediate transitions may depend only upon the discrete part of the marking. Fluid flow is determined by fluid arcs that connect timed transitions to fluid places. Each fluid arc that goes from transition t to fluid place c has associated a (possibly state $\mathcal{M}$ dependent) flow rate $r((t, c), \mathcal{M})$. If transition t is enabled in state $\mathcal{M}$, then fluid flows across the arc at $r((t, c), \mathcal{M})$ fluid units per time units. If the arc is directed from the transition to the place, then the fluid enters the fluid place. If the arc is directed from the place to the transition, fluid flows out of the place. Standard arcs are represented by standard arrows, while fluid arcs are represented as a double line, to suggest a pipe.

FSPN models are analyzed by writing the equations of the underlying stochastic processes and then by solving them numerically. The equations are written by decoupling the discrete part of the model from the continuous one.

Some extensions to the basic FSPN formalism and the problem of simulation of FSPN have been addressed in [9]. In this paper the formalism has been extended to consider fluid impulses associated with both immediate and timed transitions firings, and to allow guards associated with both timed and immediate transitions, dependent on both fluid levels and on the discrete part of the marking. Although interesting, these extensions make the models analyzable only through simulation. Simulation of FSPN, however, is not an easy task since the underlaying stochastic process becomes a non-homogenous Markov process. The problem is solved in the paper using a technique called *thinning* [25].

The main advantage of this formalism lies in the ability to write equations, wihch can describe the behavior of the model (at least for the original version). Its major drawback is the lack of fluid conservation. This means that if a transition connects two fluid places, one with an input fluid arc and the other with an output fluid arc (See Figure 1a)), the transition does not actually transfer fluid from one place to the other. If the source place becomes empty, the destination place continues to fill, making the model represented in Figure 1a) virtually equivalent to the one shown in Figure 1b). This problem may lead to models whose graphical appearance may be misinterpreted from unexperienced users.

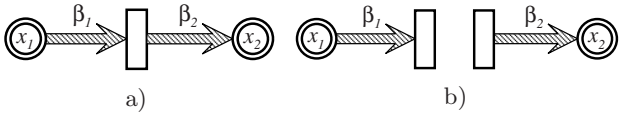


Fig. 1. The fluid conservation problem: a) an ambiguous model b) an equivalent model

Second Order Fluid Stochastic Petri Nets. Second order fluid stochastic Petri nets have been introduced by K. Wolter in [38]. Those models are an extension of the FSPN formalism described in Section 3. In second order FSPN, fluid does not flow at a constant rate, but at randomly variable rate. The flow rate is specified by a *mean flow rate* and a *flow variance*. These kind of models are called *second order* models since their solution involves the solution of a system of second order partial differential equations. In second order FSPN, each fluid arc has associated a mean flow rate and a variance. Then the potential flow rate in each discrete state is computed as the sum of the flow rate of the input arcs that connect enabled timed transitions to that place, minus the sum of the flow rate of the output arcs that connect that place to enabled timed transitions. The potential variance of the state is computed as the sum of the variance of all the fluid arcs that connect that place to enabled timed transitions, regardless of the direction of the arc.

Second order FSPNs have been extended in [39] to include fluid jumps. In this extension, each time a timed transition fires, a random amount of fluid may be added or removed from some fluid place. This is graphically represented by connecting timed transitions to fluid places with standard arcs. Since fluid places may be bounded, two jump policies exist: *force jump* and *preserve*. Using the first policy, if a jump would lead to a marking outside the boundary, it will be stopped at that boundary. Using the preserve policy, jumps that may lead the fluid levels outside the bounds are inhibited. In this way, fluid jumps after a transition firing will occur only if their magnitude will not make the fluid levels go outside the boundary.

Another extension called *Non Markovian second order FSPNs* has been proposed in [40]. This formalism makes it possible to also use timed transitions with a non exponential firing time, as long as only one general transition is enabled in a discrete marking. This formalism merges the characteristics of the second order FSPN, with the features of the non Markovian stochastic Petri nets [17].

The main drawback of this formalism is that in practical applications flow variance is very rarely required. In practice, most of the systems that are analyzed with fluid require only deterministic flow rates. Not many applications of second order FSPNs have been investigated in the literature, and this poses some additional problems to novices who want to understand the real power of the formalism.

Continuous and Hybrid Petri Nets. Continuous Petri Nets and Hybrid Petri Nets have been defined by H. Alla and R. David. A good reference to both models can be found in [5]. The main difference between these models and the one introduced in the previous sections is that fluid is not moved by arcs, but by appropriate primitives called fluid transitions. A *Continuous Petri Nets* is a model made by only fluid places and fluid transitions. Fluid is transferred along the places by the fluid transitions. A fluid transition is enabled when there is some fluid in all its input fluid places. Thanks to the fluid transitions, fluid is conserved. This implies that the flow rate of a fluid transition may be reduced when one of its input fluid places are empty. Conflicts may arise if two fluid

transitions are connected to the same fluid place. In this case two policies have been defined. With the *priority* policy, the input flow of the common fluid place is used first to enable the highest priority fluid transition. If the input flow is enough to satisfy the requirement of the highest priority transition, the remaining part is used to satisfy the request of the second highest in rank and so on. With the *sharing* policy, the input flow is shared among the output transitions, in proportion with their flow rates. In this way, all fluid transitions connected to a fluid place are always enabled, as long as there is some fluid flowing into their input fluid places.

Hybrid Petri Nets combine the power of CPN with standard Petri nets. HPN have discrete places, continuous places, discrete transitions and fluid transitions. Discrete transitions may be enabled by both discrete and continuous places, and can transfer tokens from discrete places and fluid from continuous places when they fire. Discrete transitions can convert fluid to tokens (by connecting a fluid place with an input arc and a discrete place with an output arc) and vice versa. Continuous transitions may be enabled by discrete and continuous places as well, but there is a limitation. If a continuous transition is connected with an input arc to a discrete place, it must also be connected with an output arc of the same weight to the same discrete place. This is required to "preserve the token", otherwise the fluid transition will consume a discrete quantity which is not possible.

From the modeler point of view, HPN and CPN are more easier to use, since the preservation of the fluid that enters a fluid transition is something that a graph like the one presented in Figure 1a) suggests. On the other end, requirements such as the conservation of tokens, makes the models defined with HPN a bit more complex to draw than the ones written with FSPN. For example, in Figure 2 a production / consumer model is presented using both the FSPN and the HPN formalisms. As we can see, HPN requires two more transitions and four more arcs than does the FSPN. Many applicative examples exist in the literature. In [5] a short review is made, by presenting HPNs used to model a microchip production system, a production line and a water supply system. However, the main disadvantage of the CPN and HPN approaches is that they can be analyzed only by simulation. Analytical techniques can be used only in very limited cases, even if some studies in this direction have been carried out (see for example [26]).

Fluid and diffusion models. Another class of interesting fluid based approaches is the one in which queues and buffers are approximated by fluid. In the work done by D. Mitra in [7], a producer / consumer model, with a fixed number of producers and a single consumer, with an intermediate fluid buffer is presented. In the proposed model, a Markov chain describes the state of the sources (i.e. it counts how many producers are active). In each state an input rate is computed based on the number of active producers. The difference between this value and the output rate of the consumer gives an instantaneous rate of change of the fluid buffer. A system of differential equation is then written to completely characterize the model. The proposed system is solved by

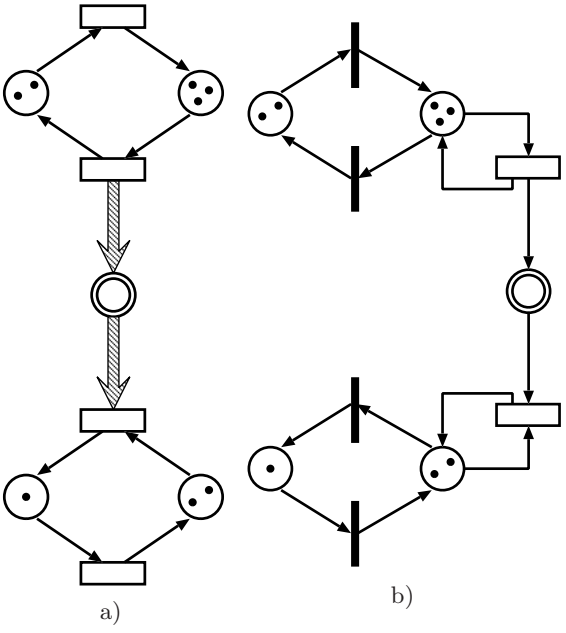


Fig. 2. A model of a producer / consumer system: a) FSPN b) HPN

computing the eigenvalues and eigenvectors of a matrix. Several techniques to obtain these eigenvalues in various cases are presented in the paper, together with some asymptotic results.

In [28] these results are extended to the case of multiple consumers.In this paper, the state of the consumers is also expressed as the state of another Markov chain. The global state of the model is defined as the Kronecker product of the two Markov chains (the one which represents the consumers and the one corresponding to the producers). In [13] the results are further extended to consider the case of different classes of sources, finite buffer, and production rates dependent on buffer occupancy. This class of models is used to describe a rate-based congestion control of a high speed network with loss priorities. The key features of these works are the abilities to characterize the stochastic process that describes the model, and to present some specific but efficient analytical solution techniques. General fluid models, where no particular structure is considered for the Markov chain that governs the fluid flow, have been addressed in several papers. A particularly interesting solution algorithm, based on Taylor expansion, has been presented in [34]. The advantages of that approach is that it does not require the computation of matrix eigenvalue and eigenvector (a task that can present many numerical problems).

Diffusion models, on the other hand, are models where the fluid flow is not considered constant in a given state, but it is defined by a mean flow rate and a variance. These models are often referred to as second order models, since they require the solution of a system of second order differential equations.

Usually these models describe the whole system using only continuous quantities. Discrete states are not considered and discrete quantities are approximated by continuous ones. The theory behind these approaches is based on the central limit theorem: a sum of a large number of random variables can be approximated by a Normal distribution characterized by a mean that corresponds to sum of the means of the various random variables, and by a variance that corresponds to the sum of the various variances. The normal distribution can then be approximated by the solution of a diffusion process that is expressed through a second order differential equation. Models are expressed as standard queuing networks with general service time distributions. Only the first two moments of these distributions (i.e. their mean and their variance) are required to find the parameters for the diffusion equation. In order to better describe the process, special boundary conditions are included. A barrier with an instantaneous jump is used to characterize the phase in which a service center is empty. A good reference to these kinds of models can be found in [11]. This approach has been extended in order to consider finite capacity queues and complex server policies. It has been used to study several applicative problems, such as cellular telephone cells or ABR traffic sources.

4 Formalisms

Continuous quantities have been introduced in performance models in many flavors. Many high-level and low-level performance evaluation formalisms have been developed to deal with continuous quantities. In this work we will consider:

- Reward Models (RM),
- Fluid Models (FM), and
- Fluid Stochastic Petri Nets (FSPN).

4.1 Reward Models

A **Reward Model** is a Markov chain in which each state has associated a positive quantity called *Reward Rate*. A continuous variable, takes into account all the Reward accumulated over a time interval. This quantity grows proportionally with the time spent in a state and with the corresponding reward rate. One of the key aspect of reward models, is that the accumulated reward is unbounded.

The Markov Chain that governs the reward accumulation is called the *underlaying Markov Chain*, and is described by a generator matrix $\mathbf{Q}$, whose element q_{ij} defines the transition from state i to state j, as in any other Markov Chain:

$$q_{ij} = \lim_{\Delta t \to 0} \frac{P\{S(t+\Delta t) = j | S(t) = i\}}{\Delta t}, \text{ for } i \neq j$$
$$q_{ij} = -\sum_{k \neq i} q_{ik}, \text{ for } i = j,$$

where $S(t)$ is the state of the underlaying Markov chain at time t.

The *reward rate* of the state i is denoted by r_i, $r_i \geq 0$. This quantity describes the rate at which the accumulated reward grows when the underlaying Markov chain is in state i. Reward rates are collected in a diagonal matrix, $\mathbf{R}$, whose elements R_{ij} are such that:

$$\begin{aligned} R_{ij} &= 0, \quad \text{for } i \neq j, \\ R_{ij} &= r_i, \quad \text{for } i = j. \end{aligned}$$

We denote with $X(t)$ the total reward accumulated until time t, and we set $X(0) = 0$. If we know the evolution of the underlaying Markov chain (that is $S(t)$), then we can compute $X(t)$ as follows:

$$X(t) = \int_0^t r_{S(u)}\ du.$$

The fact that the reward rates are always positive, and that the accumulated reward is unbounded, greatly simplifies the analytical description of the systems. Many efficient techniques exists in the literature to analyze Reward Models [32,12,24,29,31].

4.2 Fluid Models

Fluid Models are an extension of Reward Models. Various definition of fluid models exists, and they will be fully addressed in section 5. Here we will put just a general presentation of the main formalism. As RMs, FMs are characterized by an underlaying Markov Chain, defined by matrix $\mathbf{Q}$, and a reward matrix $\mathbf{R}$. The main difference with respect to RM, is that in FM the rate associated to each state (called in this case *flow rate or drift*) can be positive, negative or zero. Usually, the accumulated reward is called *Fluid Level*, since the continuous value of the reward can be used to represent the level of fluid contained in a reservoir. The second main difference between FMs and RMs, is that in a FM the fluid level has at least a lower bound at zero, and may also have an upper bound at a fixed positive value.

Even if the differences between FMs and RMs seem to be negligible at first sight, FMs are much more complex to be analyzed. The presence of boundaries and negative rates, imposes the introduction of boundary conditions in the equations that describe the evolution of the system. These conditions reduce the applicability of analytical results, and makes the solution much more complex. However, from a modeling point of view, the introduction of negative rates and bounds allows the characterization of a larger set of interesting systems, which could not be analyzed by simple RMs. FMs can be used to approximate large buffers with continuous quantities, making thus the formalism well suited to analyze high speed communication and production systems.

4.3 Fluid Stochastic Petri Nets

A **Fluid Stochastic Petri Net** (**FSPN**) is an extension of an ordinary Stochastic Petri Net, capable of incorporating continuous quantities. Other similar extensions with minor differences are: *Continuous Petri Nets* and *Hybrid Petri Nets* [5]. In this work we will not consider such formalisms, and we will present only the basic formalism, intended for stochastic analysis. Several extensions have been considered to allow the description of more complex model, which however can only be solved using simulation [10].

Formally, a FSPN is a tuple $\langle \mathcal{P}, \mathcal{T}, \mathcal{A}, B, F, W, R, M_0 \rangle$, where:

- $\mathcal{P}$ is the set of places, partitioned into a set of discrete places $\mathcal{P}_d = \{p_1, \ldots, p_{|\mathcal{P}_d|}\}$ and a set of continuous places $\mathcal{P}_c = \{c_1, \ldots, c_{|\mathcal{P}_c|}\}$ (with $\mathcal{P}_d \cap \mathcal{P}_c = \emptyset$ and $\mathcal{P}_d \cup \mathcal{P}_c = \mathcal{P}$). The discrete places may contain tokens (the number of tokens in a discrete place is a natural number), while the marking of a continuous place is a non negative real number that we call the fluid level. In the graphical representation, a discrete place is drawn as a single circle while a continuous place is signified by two concentric circles. The complete state (marking) of a FSPN is described by a pair of vectors $M = (\boldsymbol{m}, \boldsymbol{x})$, where the vector $\boldsymbol{m}$, of dimension $|\mathcal{P}_d|$ is the marking of the discrete part of the FSPN and the vector $\boldsymbol{x}$, of dimension $|\mathcal{P}_c|$, represents the fluid levels in the continuous places (with $x_l \geq 0$ for any $c_l \in \mathcal{P}_c$). We use $\mathcal{S}$ to denote the partly discrete and partly continuous state space. In the following we denote by $\mathcal{S}_d$ and $\mathcal{S}_c$ the discrete and the continuous component of the state space respectively. The marking $M = (\boldsymbol{m}, \boldsymbol{x})$ evolves in time. We can imagine the marking M at time τ as the stochastic marking process $\mathcal{M}(\tau) = \{(\boldsymbol{m}(\tau), \boldsymbol{x}(\tau)), \tau \geq 0\}$.
- $\mathcal{T}$ is the set of transitions partitioned into a set of stochastically timed transitions $\mathcal{T}_e$ and a set of immediate transitions $\mathcal{T}_i$ (with $\mathcal{T}_e \cap \mathcal{T}_i = \emptyset$ and $\mathcal{T}_e \cup \mathcal{T}_i = \mathcal{T}$). A timed transition $T_j \in \mathcal{T}_e$ is drawn as a rectangle and has an instantaneous firing rate associated with it. An immediate transition $t_h \in \mathcal{T}_i$ is signified by a thin bar and has constant zero firing time.
- $\mathcal{A}$ is the set of arcs partitioned into three subsets: $\mathcal{A}_d$, $\mathcal{A}_h$ and $\mathcal{A}_c$. The subset $\mathcal{A}_d$ contains the discrete arcs which can be seen as a function[1] $A_d : ((\mathcal{P}_d \times \mathcal{T}) \cup (\mathcal{T} \times \mathcal{P}_d)) \to I\!N$. The arcs $\mathcal{A}_d$ are drawn as single arrows. The subset $\mathcal{A}_h$ contains the inhibitor arcs, $A_h : (\mathcal{P}_d \times \mathcal{T}) \to I\!N$. These arcs are drawn with a small circle at the end. The definitions of ${}^\bullet t_j$, $t_j^\bullet$, and ${}^\circ t_j$ involve only discrete places and are the same as for the standard GSPNs.

  The subset $\mathcal{A}_c$ define the continuous arcs. These arcs are drawn as double arrows to suggest a pipe. $\mathcal{A}_c$ is a subset of $(\mathcal{P}_c \times \mathcal{T}_e) \cup (\mathcal{T}_e \times \mathcal{P}_c)$, i.e., a continuous arc can connect a fluid place to a timed transition or it can connect a timed transition to a fluid place.
- The function $B : \mathcal{P}_c \to I\!R^+ \cup \{\infty\}$ describes the fluid upper boundaries on each continuous place. This boundary has no effect when it is set to infinity. From this if follows that $\forall\, M = (\boldsymbol{m}, \boldsymbol{x}) \in \mathcal{S}$ and $c_l \in \mathcal{P}_c$, $0 \leq x_l \leq B(c_l)$. Each fluid place has an implicit lower boundary at level 0.

[1] Note that when the arcs are defined as a function we use uppercase symbols.

- The firing rate function F is defined for timed transitions $\mathcal{T}_e$ so that $F : \mathcal{T}_e \times \mathcal{S} \rightarrow I\!R^+$. Therefore, a timed transition T_j enabled at time τ in a discrete marking $\boldsymbol{m}(\tau)$ with fluid level $\boldsymbol{x}(\tau)$, may fire with rate $F(T_j, \boldsymbol{m}(\tau), \boldsymbol{x}(\tau))$, that is:
$$\lim_{\Delta\tau \to 0} Pr\{T_j \text{ fires in } (\tau, \tau+\Delta\tau) | \mathcal{M}(\tau) = (\boldsymbol{m}(\tau), \boldsymbol{x}(\tau))\} = F(T_j, \boldsymbol{m}, \boldsymbol{x})\Delta\tau$$
We also use as a short hand notation $F(T_j, M)$, where $M = (\boldsymbol{m}, \boldsymbol{x})$).
- The weight function W is defined for immediate transitions $\mathcal{T}_i$ such that $W : \mathcal{T}_i \times \mathcal{S}_d \rightarrow I\!R^+$. Note that the firing rates for timed transitions may be dependent both on the discrete and the continuous part of the marking, while the weights for immediate transitions may only be dependent on the discrete part.
- The function $R : \mathcal{A}_c \times \mathcal{S} \rightarrow I\!R^+ \cup \{0\}$ is called the *flow rate function* and describes the marking dependent flow of fluid across the input and output continuous arcs connecting timed transitions and continuous places.
- The initial state of the FSPN is denoted by the pair $M_0 = (\boldsymbol{m_0}, \boldsymbol{x_0})$.

Figure 3 visually represents the discrete primitives of a FSPN (that are identical to their GSPN counterparts), and Figure 4 shows the continuous primitives of the formalism.

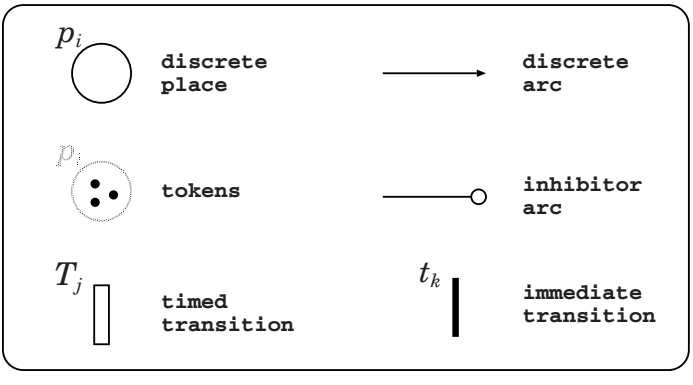


Fig. 3. Discrete primitives

The role of the previous sets and functions will be clarified by providing the enabling and firing rules.

Let us denote by m_i the i-th component of the vector $\boldsymbol{m}$, i.e., the number of tokens in place p_i when the discrete marking is $\boldsymbol{m}$. We say that a transition $t_j \in \mathcal{T}$ (no matter whether t_j is an immediate or timed transition) has concession in marking $M = (\boldsymbol{m}, \boldsymbol{x})$ iff $\forall p_i \in {}^\bullet t_j, m_i \geq A_d(p_i, t_j)$ and $\forall p_i \in {}^\circ t_j, m_i < A_h(p_i, t_j)$. If an immediate transition has concession in $M = (\boldsymbol{m}, \boldsymbol{x})$, it is said to be enabled and the marking is vanishing. Otherwise, the marking is tangible and any timed transition with concession is enabled in it. Note that the previous definition is exactly the one of standard GSPNs [4], i.e., the concession and the enabling

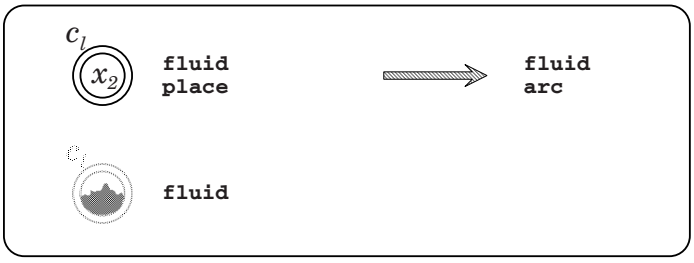


Fig. 4. Continuous primitives

conditions depend only on the discrete part of the FSPN. Let $\mathcal{E}(M)$ denote the set of enabled transitions in marking $M = (\boldsymbol{m}, \boldsymbol{x})$, we have that $\mathcal{E}(M) = \mathcal{E}(M')$, for any marking $M' = (\boldsymbol{m}, \boldsymbol{x'})$.

The stochastic evolution of the discrete part of the FSPN in a tangible marking is governed by a race [3]. In a vanishing marking, instead, the weights are used to decide which transition should fire according to the standard rules for GSPNs [4]. Let us see how enabled transitions may influence the continuous part of the marking. Each continuous arc that connects a fluid place $c_l \in \mathcal{P}_c$ to an enabled timed transition $T_j \in \mathcal{T}_e$ (resp. an enabled transition T_j to a fluid place c_l), causes a "change" in the fluid level of place c_l. Let $\mathcal{M}(\tau)$ be the marking process, i.e., $\mathcal{M}(\tau) = M_i$ if at time τ the marking of the FSPN is $M_i = (\boldsymbol{m_i}, \boldsymbol{x_i})$. Thus, when the FSPN marking is $\mathcal{M}(\tau)$ fluid can leave place $c_l \in \mathcal{P}_c$ along the arc $(c_l, T_j) \in \mathcal{A}_c$ at rate $R((c_l, T_j), \mathcal{M}(\tau))$ and can enter the continuous place c_l at rate $R((T_j, c_l), \mathcal{M}(\tau))$ for each (timed) transition T_j enabled in $\mathcal{M}(\tau)$. The potential rate of change of fluid level for the continuous place c_l in marking $\mathcal{M}(\tau)$ is:

$$r_l^p(\mathcal{M}(\tau)) = \sum_{T_j \in \mathcal{E}(\mathcal{M}(\tau))} R((T_j, c_l), \mathcal{M}(\tau)) - R((c_l, T_j), \mathcal{M}(\tau)).$$

We require that for every discrete marking $\boldsymbol{m}$ and continuous arc (c_l, T_j) (resp. (T_j, c_l)),that the rate $R((c_l, T_j), (\boldsymbol{x}, \boldsymbol{m}))$ (resp. $R((T_j, c_l), (\boldsymbol{x}, \boldsymbol{m}))$) be a piecewise continuous function of $\boldsymbol{x}$.

Now let $X_l(\tau)$ be the fluid level at time τ in a continuous place $c_l \in \mathcal{P}_c$. The fluid level in each continuous place c_l can never become negative or exceed the bound $B(c_l)$, so the (actual) rate of change over time, τ, when the marking is $\mathcal{M}(\tau)$, is

$$r_l(\mathcal{M}(\tau)) = \frac{dX_l(\tau)}{d\tau} = \begin{cases} r_l^p(\mathcal{M}(\tau)) & \text{if } X_l(\tau) = 0 \text{ and } r_l^p(\mathcal{M}(\tau)) \geq 0 \\ r_l^p(\mathcal{M}(\tau)) & \text{if } X_l(\tau) = B(c_l) \text{ and } r_l^p(\mathcal{M}(\tau)) < 0 \\ 0 & \text{if } X_l(\tau) = 0 \text{ and } r_l(\mathcal{M}(\tau)) < 0 \\ 0 & \text{if } X_l(\tau) = B(c_l) \text{ and } r_l^p(\mathcal{M}(\tau)) > 0 \\ r_l^p(\mathcal{M}(\tau)) & \text{if } 0 < X_l(\tau) < B(c_l) \text{ and } r_l^p(\mathcal{M}(\tau^-)) r_l^p(\mathcal{M}(\tau^+)) \geq 0 \\ 0 & \text{if } 0 < X_l(\tau) < B(c_l) \text{ and } r_l^p(\mathcal{M}(\tau^-)) r_l^p(\mathcal{M}(\tau^+)) < 0. \end{cases} \quad (1)$$

The first two cases of the previous equation concern situations when $X_l(\tau) = 0$ (resp. $X_l(\tau) = B(c_l)$) and the potential rate is $r_l^p(\mathcal{M}(\tau)) \geq 0$ (resp. $r_l^p(\mathcal{M}(\tau)) < 0$). In both cases the actual rate is equal to the potential rate. The third and the fourth cases prevent the fluid level from overcoming the lower and the upper boundaries. The last two cases require a deeper explanation (a reference for a complete discussion of these aspects is [13]). As it has been assumed in [21], the flow rate function $R(\cdot,\cdot)$ is a piecewise continuous function of the continuous part of the marking. The meaning of the last case is that a sign change (from $+$ to $-$) in $r_l^p(\mathcal{M}(\tau))$ will "trap" $X_l(\tau)$ in a state in which it will be constant. With this assumption, the analysis of the stochastic process $\mathcal{M}(\tau)$ is simplified (see [13] for a discussion on this type of situation). The fifth case, which is the most common one, accounts for the fact that there is no sign change from $+$ to $-$ in $r_l^p(\mathcal{M}(\tau))$ and hence the actual rate is equal to the potential rate.

Fluid Stochastic Petri Nets are analyzed by transforming them into equivalent Fluid Models. If the FSPN has a single fluid place, then standard FM can be applied. If the FSPN has more than one fluid place, then special FM with multiple continuous variables must be used.

We will begin by describing how to compute the *infinitesimal generator* $\boldsymbol{Q}$ of the equivalent FM. Since fluid arcs arcs do not change the enabling condition of a transition, standard analysis techniques can be applied to the discrete marking process $\boldsymbol{m}(\tau)$ [4]. These techniques split the discrete state space into two disjoint subsets; called respectively, the *tangible marking* set and the *vanishing marking* set. Since the process spends no time in vanishing markings, they can be removed and their effect can be included in the transitions between tangible markings. From this point on, we will consider only tangible markings. In GSPNs, the underlying stochastic process is a CTMC, whose infinitesimal generator is a matrix $\boldsymbol{Q}$. Each entry q_{ij} represents the rate of transition from a tangible state $\boldsymbol{m}_i$ to a tangible state $\boldsymbol{m}_j$, that is:

$$q_{ij} = \sum_{T_k \in \mathcal{E}(\boldsymbol{m}_i) \mid \boldsymbol{m}_i \stackrel{T_k}{\rightarrow} \boldsymbol{m}_j} F(T_k, \boldsymbol{m}_i),$$

where $\mathcal{E}(\boldsymbol{m}_i)$ represents the set of enabled transitions in marking $\boldsymbol{m}_i$, and $\boldsymbol{m}_i \stackrel{T_k}{\rightarrow} \boldsymbol{m}_j$ means that the firing of transition T_k changes the discrete state of the system from $\boldsymbol{m}_i$ to $\boldsymbol{m}_j$.

In the FSPN model defined in [21], the firing rate of each timed transition can be made dependent on the continuous component of the state. With this extension, the infinitesimal generator matrix must be also dependent on the fluid component of the state, that is $\boldsymbol{Q}(\boldsymbol{x}) = \{q_{ij}\}$ where:

$$q_{ij}(\boldsymbol{x}) = \sum_{T_k \in \mathcal{E}(\boldsymbol{m}_i) \mid \boldsymbol{m}_i \stackrel{T_k}{\rightarrow} \boldsymbol{m}_j} F(T_k, \boldsymbol{m}_i, \boldsymbol{x}).$$

The summation considers the transition rates of all the transitions T_k that bring the net from state $\boldsymbol{m}_i$ to $\boldsymbol{m}_j$. In the standard equations that describe a CTMC,

the terms on the diagonal of the infinitesimal generator account for the probability of exiting from a state. In this case, we simply define:

$$q_{ii}(\boldsymbol{x}, \emptyset) = -\sum_{j \neq i} q_{ij}(\boldsymbol{x}). \tag{2}$$

Matrix $\boldsymbol{Q}(\boldsymbol{x})$ is equivalent to the infinitesimal generator of a CTMC, in the sense that each row sum of $\sum_{\mathbf{s} \in 2^{\mathcal{P}_c}} Q(\boldsymbol{x}, \mathbf{s})$, is equal to zero. In other words:

$$Q(\boldsymbol{x})\mathbf{1} = \mathbf{0}$$

where $\mathbf{1}$ (respectively $\mathbf{0}$) is a column vector with all the $|\mathcal{S}_d|$ components equal to 1 (resp. 0).

If we have only a single fluid place c_l, the fluid rate matrix $R(\boldsymbol{x})$, of the underlaying fluid model, can be simply computed by defining $r(i, x) = r_l(M)$, where $M = (\boldsymbol{m_i}, x)$. Then $\boldsymbol{R}(x) = \text{diag}(r(i, x))$ becomes the diagonal matrix whose components account for the actual flow rate out of the fluid place.

No boundary conditions are needed, since they are included in the definition of the potential flow rate (Equation (11)). Dirac's delta functions in the solution, represent cases where there is a non zero probability of finding the system in a particular marking (both discrete and continuous).

In [19] a new kind of fluid primitive, called Flush-out arcs has been added to the FSPN formalism.

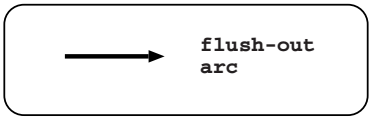


Fig. 5. Flushout arcs

Flush-out (See Figure 5 arcs are special arcs that connect fluid places to timed transition (but not timed transition to fluid places). They are drawn using thick lines. When a transition fires, the places connected with a flush-out arc are emptied in zero time.

Despite their simplicity, Flush-out Arcs can be exploited to obtain many interesting effects, like dropping the content of the transmission buffer. The underlaying stochastic model is no longer a standard Fluid Model, but it can be analyzed similarly using appropriate boundary conditions. It has been shown in [19] that FSPNs with flush-out arcs can be used to simulate Non-Markovian Stochastic Petri Nets [15].

5 Analytical Description of Fluid Models

Since the behaviour of the considered class of fluid models contain random elements they belong to the large family of stochastic processes. Stochastic

processes can be viewed as a set of random variables, which are ordered according to a parameter. In typical engineering applications the parameter represents the time and it takes value either from the natural numbers, $0, 1, 2, \ldots$, or from the set of non-negative real numbers. The former case is referred to as discrete time stochastic process and the later as continuous time stochastic process.

Also in typical engineering applications the random variables has the same support set. The characteristics of this support set is the other main feature of the stochastic process. We distinguish the following cases:

- discrete support set, e.g., the number of customers in a queue,
- continuous support set, e.g., the unfinished work in a queue,
- hybrid (continuous and discrete) support set, e.g., the unfinished work and the number of customers.

General continuous and hybrid valued stochastic processes are hard to analyze but, there are special cases which allow the application of simple analysis methods. Focusing on the hybrid valued case the simplest processes are obtained when the continuous part of the model is determined by its discrete part through a very simple function, which is the case with reward models and fluid models.

In both cases a simple function of a discrete state stochastic process governs the evolution of the continuous variable. E.g., the continuous value is increasing or decreasing at a given rate while the discrete value is constant (see Figure 6). In case of reward models this evolution is non-decreasing and unbounded, while in case of fluid models the evolution of the continuous variable is bounded.

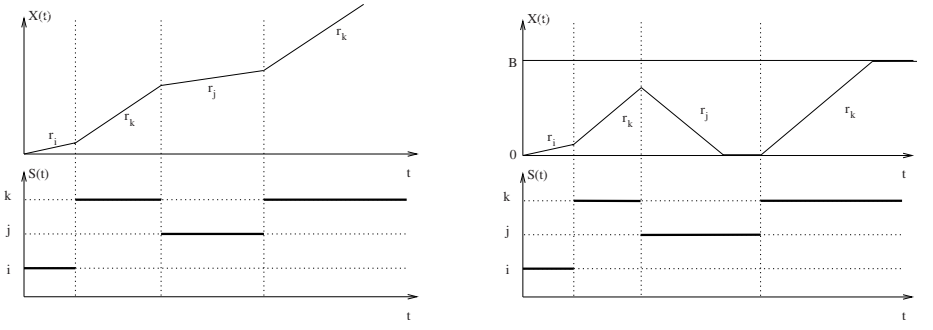


Fig. 6. Evolution of the continuous variable in reward and fluid models

Definition 1. *Markov property: A stochastic process is said to enjoy the Markov property at time t, when the future evolution of the process is independent of its past and depend only on the value of the random variable at time t.*

Those stochastic processes which enjoy the Markov property at all time are referred to as Markov processes. Continuous time Markov chains, Markov reward models, Markov fluid models are examples of Markov processes. In this chapter we focus on these Markovian cases.

5.1 Classification of Fluid Models

The following features of fluid models are used for classification:

– Buffer size:
It is commonly assumed that the minimal buffer level is 0. This way the size of the buffer determines the maximal buffer content. The two main cases are **finite buffer** and **infinite buffer**. In case of an infinite buffer the continuous quantity is only lower bounded at zero and in case of a finite buffer the continuous quantity is lower bounded at zero and upper bounded at B (Figure 7).

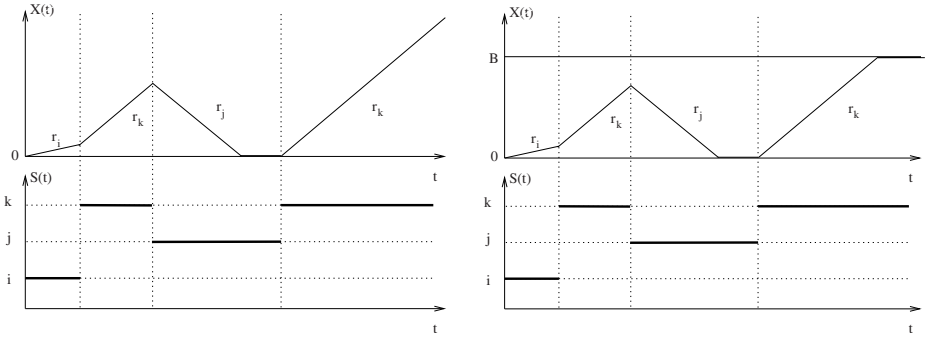


Fig. 7. Infinite and finite buffer fluid models

– Evolution of the continuous variable:
The evolution of the continuous variable depends on the value of the discrete variable, but this dependence can be of two kinds. The case when the continuous variable deterministically increases/decreases as long as the discrete variable is constant, is referred to **first order** fluid model (Figure 8). The case when the increment of the continuous variable during a period when the discrete variable is constant is a normal distributed random quantity is referred to **second order** fluid model (Figure 8).

Second order fluid models can be interpreted as the limiting process of a two dimensional random walk according to Figure 9, where the horizontal dimension represents the discrete variable and the vertical dimension represents the buffer level.

In this model the probabilities of the vertical state transitions determine the mean fluid increase for each value of the discrete variable. It can be seen that the fluid level can increase and also decrease in each column.

Reducing the time step and the granularity of the fluid level of this model to zero results in the second order fluid flow model.

– Effect of the buffer content on the discrete variable:
With this respect there are two main cases. The evolution of the discrete variable can be independent of or can depend on the instantaneous value

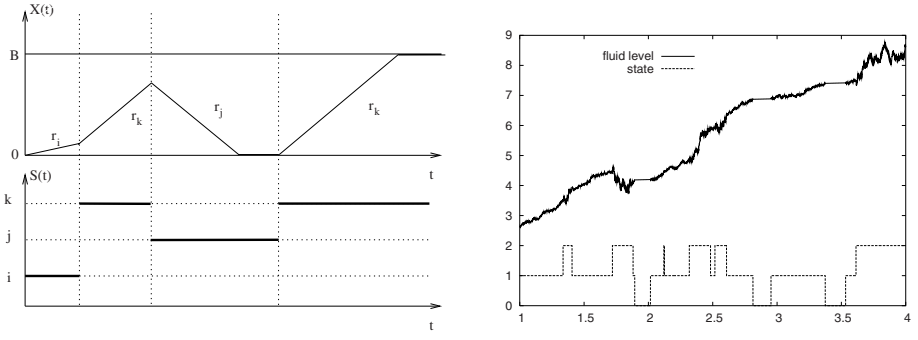


Fig. 8. First and second order fluid models

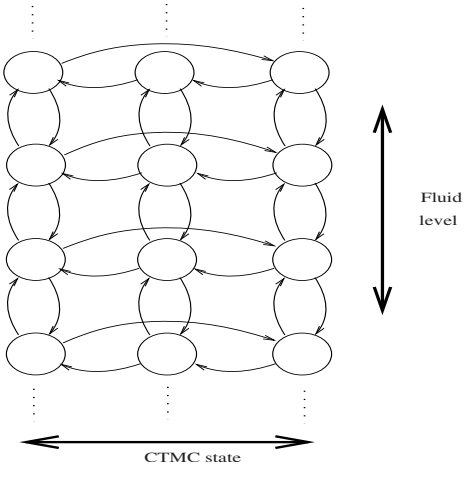


Fig. 9. Interpretation of second order fluid models

of the fluid variable. In case of a Markov fluid model, the first means that the discrete part of the model is an "independent" CTMC which modulates the fluid accumulation. In the later case there is a mutual dependence of the continuous and the discrete part of the model and it is not possible to analyze the discrete variable in isolation. The first case is also referred to as space **inhomogeneous model**, since the generator matrix of the discrete variable is constant, i.e., independent of the fluid level, while the second case is also referred to as **fluid level dependent** model.

– Behaviour of the second order model at the boundaries:
In case of first order fluid models the model behaviour is quite well defined when the fluid level reaches a boundary. When the fluid level reaches the lower boundary (empty buffer) the system must be in a state with a negative

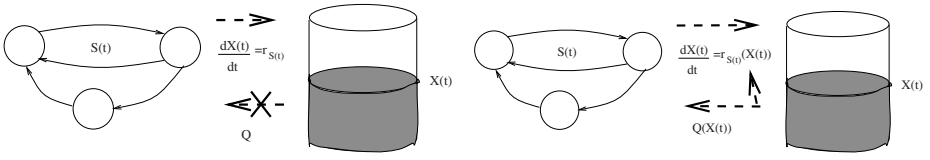


Fig. 10. Markov fluid models with independent and dependent discrete parts

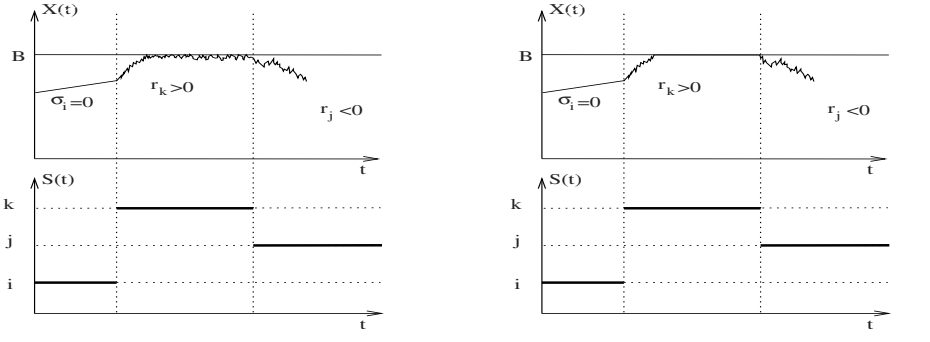


Fig. 11. Second order fluid models with reflecting and absorbing boundary behaviour

fluid rate. After reaching the lower boundary (in a state with a negative fluid rate), the buffer remains empty as long as a transition to a state with positive fluid rate takes place. The system behaviour at the upper boundary (if any) is similar.

The behaviour of second order fluid models is more complex at the boundaries. In this case we might assume a deterministic and a stochastic behaviour depending on the behaviour of the modeled system.

The deterministic boundary behaviour of second order models is more or less identical with the described boundary behaviour of the first order model. The only difference is that the fluid level can become zero also in states with non-negative drift and positive variance. This case is referred to as **absorbing boundary**, since the fluid level gets identical with the boundary for a positive amount of time (Figure 11).

The stochastic boundary behaviour of second order models is similar to the general evolution of these models between the boundaries, where the fluid level alternates randomly all over the time and does not remain constant for a non-zero time period with positive probability. In this case, the fluid level process is reflected as soon as it reaches a boundary, and this way it always remain between the boundaries with probability 1. This case is referred to as **reflecting boundary**, since the fluid level gets reflected at the boundary (Figure 11).

These boundary behaviours can be interpreted using the same random walk approximation as we used for the interpretation of the second order fluid

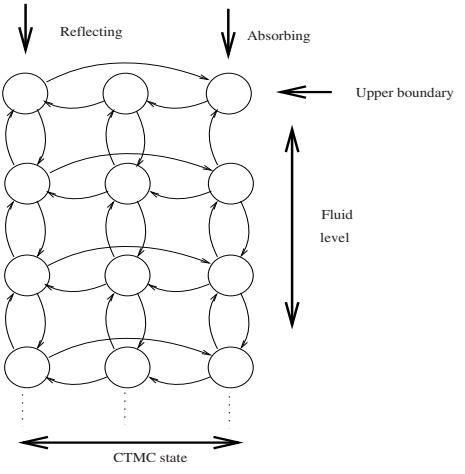


Fig. 12. Interpretation of the boundary behaviour

model. Those states exhibit absorbing boundary behaviour, where there is no vertical state transition out of a boundary state, i.e., the discrete variable must change its value to leave the given boundary state. (Figure 12). In contrast, those states exhibit reflecting behaviour, where there is a vertical transition out of the boundary state, i.e., the fluid level is alternating all over the time.

5.2 Transient Behaviour of the Fluid Level Process

In this section we study the transient behaviour of the fluid level process according to a simple to more complex approach. The simplest case is the transient analysis of *first order, infinite buffer, homogeneous* Markov fluid models. Later on we extend this model with finite buffer, second order fluid accumulation and fluid level dependency.

First order, infinite buffer, homogeneous Markov fluid models. During an infinitesimally short period of time, $(t, t+\Delta)$, while the continuous variable is i, or equivalently we also say that the (discrete part of the) system is in state i, $(S(\tau) = i, \forall \tau \in (t, t+\Delta))$, the fluid level $(X(t))$ increases at rate r_i when $X(t) > 0$:

$$X(t+\Delta) - X(t) = r_i \Delta$$

that is

$$\frac{d}{dt}X(t) = r_i \quad \text{if } S(t) = i, X(t) > 0.$$

When $X(t) = 0$ the fluid level can not decrease:

$$\frac{d}{dt}X(t) = \max(r_i, 0) \quad \text{if } S(t) = i, X(t) = 0.$$

That is

$$\frac{d}{dt}X(t) = \begin{cases} r_{S(t)} & \text{if } X(t) > 0, \\ \max(r_{S(t)}, 0) & \text{if } X(t) = 0, \end{cases}$$

where $r_{S(t)}$ denotes the fluid rate in the actual discrete state of the process.

First order, finite buffer, homogeneous Markov fluid models. When $X(t) = B$ the fluid level can not increase:

$$\frac{d}{dt}X(t) = \min(r_i, 0), \quad \text{if } S(t) = i, X(t) = B.$$

That is

$$\frac{d}{dt}X(t) = \begin{cases} r_{S(t)}, & \text{if } X(t) > 0, \\ \max(r_{S(t)}, 0), & \text{if } X(t) = 0, \\ \min(r_{S(t)}, 0), & \text{if } X(t) = B. \end{cases}$$

Second order, infinite buffer, homogeneous Markov fluid models with reflecting barrier. During a sojourn in the discrete state i ($S(t) = i$) in the sufficiently small $(t, t + \Delta)$ interval the distribution of the fluid increment $(X(t + \Delta) - X(t))$ is normal distributed with mean $r_i\Delta$ and variance $\sigma_i^2\Delta$:

$$X(t + \Delta) - X(t) = \mathcal{N}(r_i\Delta, \sigma_i^2\Delta), \quad \text{if } S(u) = i, u \in (t, t + \Delta), X(t) > 0,$$

where $\mathcal{N}(r_i\Delta, \sigma_i^2\Delta)$ denotes a normal distributed random variable with mean $r_i\Delta$ and variance $\sigma_i^2\Delta$.

If at $X(t) = 0$ the fluid process is reflected immediately in state i, it means that the time spent at the boundary has a 0 measure, and so the probability of staying at the boundary is

$$Pr(X(t) = 0, S(t) = i) = 0.$$

Second order, infinite buffer, homogeneous Markov fluid models with absorbing barrier. Between the boundaries the evolution of the process is the same as before.

When the fluid level decreases to zero in an absorbing barrier state, i, the fluid process gets stopped and the fluid level remains zero for a positive amount of time. Due to this behaviour

$$Pr(X(t) = 0, S(t) = i) > 0.$$

On the other hand, due to the absorbing property of the boundary the probability that the fluid level is close to the boundary in an absorbing state is very low,

$$\lim_{\Delta \to 0} \frac{Pr(0 < X(t) < \Delta, S(t) = i)}{\Delta} = 0.$$

Inhomogeneous (fluid level dependent), first order, infinite buffer Markov fluid models. The evolution of the fluid level differs due to the fluid level dependency of the fluid rate $r_i(x)$, where x is the fluid level. This way the fluid level changes as:

$$\frac{d}{dt}X(t) = \begin{cases} r_{S(t)}(X(t)), & \text{if } X(t) > 0, \\ \max(r_{S(t)}(X(t)), 0), & \text{if } X(t) = 0\ . \end{cases}$$

The evolution of the discrete part also depends on the fluid level. The detailed discussion of this dependence is delayed to the next subsection.

5.3 Transient Description of Fluid Models

The aim of this section is to derive partial differential equations representing the evolution of Markov fluid models in time. To this end we introduce the following notations:

- $\pi_i(t) = Pr(S(t) = i)$ – state probability,
- $u_i(t) = Pr(X(t) = B, S(t) = i)$ – buffer full probability,
- $\ell_i(t) = Pr(X(t) = 0, S(t) = i)$ – buffer empty probability,
- $p_i(t, x) = \lim_{\Delta \to 0} \frac{1}{\Delta} Pr(x < X(t) < x + \Delta, S(t) = i)$ – fluid density.

Based on these notations the total probability law for time t gives

$$\pi_i(t) = \ell_i(t) + u_i(t) + \int_x p_i(t, x)dx. \tag{3}$$

In the simplest considered case the evolution of the discrete variable is independent of the fluid level, hence the discrete part of the model is a continuous time Markov chains (CTMC).

Continuous time Markov chains. We start with the transient behaviour of CTMCs and based on that we extend the analysis to the various fluid models. The transient behaviour of CTMCs is characterized by the transition rates that are determined by the transition probabilities as follows

$$\lim_{\Delta \to 0} \frac{Pr(S(t + \Delta) = j | S(t) = i)}{\Delta} = q_{ij}\ . \tag{4}$$

The commonly applied forward argument to analyze Markovian stochastic processes is based on the analysis of the short term behaviour of the process. We can say that a CTMC which is in state i at time t can do 3 different things in the $(t, t + \Delta)$ interval:

- no state transition:
  The process can remain in state i during the whole period with probability $1 - \sum_{j, j \neq i} q_{ij}\Delta + \sigma(\Delta)$, where $q_{ij}\Delta$ is the probability of a state transition from state i to j in the $(t, t+\Delta)$ interval and $\sigma(\Delta)$ is a small error term that

quickly vanishes as Δ tends to zero, i.e., $\lim_{\Delta \to 0} \sigma(\Delta)/\Delta = 0$. Introducing the notation, $q_{ii} = -\sum_{j,j \neq i} q_{ij}$, we can write the probability of no state transition in the $(t, t+\Delta)$ interval as $1 + q_{ii}\Delta + \sigma(\Delta)$. Note that q_{ii} is negative.

- one state transition:
  The process can have a state transition from i to j during the $(t, t+\Delta)$ interval with probability $q_{ij}\Delta + \sigma(\Delta)$.
- more than one state transitions:
  The probability of having more than one state transitions in a short interval is quickly vanishes as Δ tends to zero, it is $\sigma(\Delta)$.

Based on these three options we can evaluate the probability of being in state i at time $t+\Delta$ ($\pi_i(t+\Delta)$) as a function of the probability of being in the various states at time t ($\pi_j(t)$):

$$\pi_i(t+\Delta) = \Big(1 + q_{ii}\Delta + \sigma(\Delta)\Big)\pi_i(t) + \sum_{j,j \neq i} \Big(q_{ji}\Delta + \sigma(\Delta)\Big)\pi_j(t) + \sigma(\Delta).$$

A $\sigma(\Delta)$ function multiplied with a bounded function ($0 \leq \pi_i(t) \leq 1$) remains to be a $\sigma(\Delta)$ function as well as the finite sum of such functions. Using this we can rearrange the expression to

$$\pi_i(t+\Delta) - \pi_i(t) = q_{ii}\Delta\pi_i(t) + \sum_{j,j \neq i} q_{ji}\Delta\pi_j(t) + \sigma(\Delta) = \sum_j q_{ji}\Delta\pi_j(t) + \sigma(\Delta).$$

Dividing both sizes by Δ and making the $\Delta \to 0$ limit we have

$$\frac{\pi_i(t+\Delta) - \pi_i(t)}{\Delta} = \sum_j q_{ji}\pi_j(t) + \frac{\sigma(\Delta)}{\Delta}$$

and

$$\frac{d\pi_i(t)}{dt} = \sum_j \pi_j(t) q_{ji} \ . \tag{5}$$

(5) is the differential equation describing the transient behaviour of CTMCs. We apply the same forward approach to evaluate the transient behaviour of Markov fluid models.

First order, infinite buffer, homogeneous Markov fluid models. We perform the same analysis for the fluid density using these 3 possible events during the $(t, t+\Delta)$ interval. For simplicity we neglect the unnecessary $\sigma(\Delta)$ terms.

If $S(t+\Delta) = i$, then during the $(t, t+\Delta)$ interval the CTMC

- stays in i and increases the fluid level with $r_i\Delta$ with probability $1 + q_{ii}\Delta$,

– moves from k to i and changes the fluid level with $\mathcal{O}(\Delta)$ with probability $q_{ki}\Delta$, where $\mathcal{O}(\Delta)$ is a function which vanishes as Δ tends to zero, i.e., $\lim_{\Delta\to 0}\mathcal{O}(\Delta) = 0$ (in this particular case the change of the fluid level is between $r_i\Delta$ and $r_j\Delta$),
– has more than 1 state transition with probability $\sigma(\Delta)$.

Considering these three cases we can express the fluid density at time $t+\Delta$ as a function of the fluid density at time t:

$$\begin{aligned} p_i(t+\Delta,x) \;= (1+q_{ii}\Delta)\; & p_i(t, x-r_i\Delta)+ \\ & \sum_{k\in\mathcal{S},k\neq i} q_{ki}\Delta\; p_k(t,x-\mathcal{O}(\Delta))+ \\ & \sigma(\Delta)\;. \end{aligned}$$

Rearranging the terms, dividing both sides by Δ and making the $\Delta\to 0$ limit gives

$$\begin{aligned} p_i(t+\Delta,x) - p_i(t,x-r_i\Delta) = \\ \sum_{k\in\mathcal{S}} q_{ki}\Delta\; p_k(t,x-\mathcal{O}(\Delta)) + \sigma(\Delta)\;, \end{aligned}$$

$$\begin{aligned} \frac{p_i(t+\Delta,x)-p_i(t,x)}{\Delta} + r_i\frac{p_i(t,x)-p_i(t,x-r_i\Delta)}{r_i\Delta} = \\ \sum_{k\in\mathcal{S}} q_{ki}\; p_k(t,x-\mathcal{O}(\Delta)) + \frac{\sigma(\Delta)}{\Delta}\;, \end{aligned}$$

$$\boxed{\frac{\partial}{\partial t}p_i(t,x) + r_i\frac{\partial}{\partial x}p_i(t,x) = \sum_{k\in\mathcal{S}} q_{ki}\; p_k(t,x)\;.} \tag{6}$$

(6) is the basic partial differential equation describing the transient behaviour of Markov fluid models. Indeed this equation describes the process behaviour during the period while the fluid level is between the boundaries.

The model behaviour at the boundaries can be obtained by the same forward argument. If $r_i > 0$, than the fluid level increases in state i, which means that the buffer cannot be empty in state i, i.e., $\ell_i(t) = Pr(X(t) = 0, S(t) = i) = 0$.

If $r_i \leq 0$, we can consider the same 3 cases for the $(t, t+\Delta)$ interval:

– If there is no state transition the fluid level is zero at $t+\Delta$ if it was zero at t or if it was between 0 and $-r_i\Delta$ at t.
– If there is one state transition in the interval, the fluid level was zero at t or if it was between 0 and $\mathcal{O}(\Delta)$ (between $-r_i\Delta$ and $-r_k\Delta$) at t.
– The case of having more than one state transitions in the interval, is treated in the same way as before.

$$\begin{aligned}\ell_i(t+\Delta) \;=\; & \\ & (1+q_{ii}\Delta)\left(\ell_i(t)+\underbrace{\int_0^{-r_i\Delta} p_i(t,x)dx}_{*}\right)+ \\ & \sum_{k\in\mathcal{S},k\neq i} q_{ki}\Delta\left(\ell_k(t)+\underbrace{\int_0^{\mathcal{O}(\Delta)} p_k(t,x)dx}_{\mathcal{O}(\Delta)}\right)+ \\ & \sigma(\Delta)\ .\end{aligned}$$

When $x \leq -r_i\Delta$, then using the first elements of the Taylor series of $p_i(t,x)$, we have

$$p_i(t,x) = p_i(t,0) + xp_i'(t,0) + \sigma(\Delta)\ ,$$

and substituting it into the previous expression we obtain

$$\begin{aligned} * &= \int_0^{-r_i\Delta} p_i(t,x)dx \\ &= \int_0^{-r_i\Delta} p_i(t,0)dx + \int_0^{-r_i\Delta} xp_i'(t,0)dx + \int_0^{-r_i\Delta}\sigma(\Delta)dx \\ &= -r_i\Delta\ p_i(t,0) \quad + \underbrace{\frac{(-r_i\Delta)^2}{2}\ p_i'(t,0)}_{\sigma(\Delta)} \quad + \underbrace{(-r_i\Delta)\ \sigma(\Delta)}_{\sigma(\Delta)}\ . \end{aligned}$$

From which we can calculate the differential equation for the empty buffer probability using the same steps as before:

$$\begin{aligned}\ell_i(t+\Delta) \;&=(1+q_{ii}\Delta)\ \left(\ell_i(t)\ -r_i\Delta p_i(t,0)+\sigma(\Delta)\ \right)+ \\ &\qquad \sum_{k\in\mathcal{S},k\neq i} q_{ki}\Delta\ (\ell_k(t)+\mathcal{O}(\Delta))+\sigma(\Delta)\ ,\end{aligned}$$

$$\begin{aligned}\ell_i(t+\Delta)-\ell_i(t) \;&= q_{ii}\Delta\ \ell_i(t) - r_i\Delta p_i(t,0)+ \\ &\qquad \sum_{k\in\mathcal{S},k\neq i} q_{ki}\Delta\ (\ell_k(t)+\mathcal{O}(\Delta))+\sigma(\Delta)\ ,\end{aligned}$$

$$\begin{aligned}&\frac{\ell_i(t+\Delta)-\ell_i(t)}{\Delta} \;= \\ &\quad -r_i\ p_i(t,0)+\sum_{k\in\mathcal{S}} q_{ki}\ (\ell_k(t)+\mathcal{O}(\Delta))+\frac{\sigma(\Delta)}{\Delta}\ ,\end{aligned}$$

$$\boxed{\frac{d}{dt}\ell_i(t) \;= -r_i\ p_i(t,0)+\sum_{k\in\mathcal{S}} q_{ki}\ \ell_k(t)\ .} \tag{7}$$

Having these expressions we can conclude the transient description of first order, infinite buffer, homogeneous Markov fluid models. The fluid density is governed by (6) while the empty buffer probability is $\ell_i(t) = 0$ if $r_i > 0$ and (7) if $r_i \leq 0$. There is no simple symbolic solution to this set of differential equations. When the initial condition of the fluid model is known, it can be solved using numerical techniques. The solution has to fulfill the following equations:

$$\int_0^\infty p_i(t,x)dx + \ell_i(t) = \pi_i(t) \ . \tag{8}$$

$$\pi_i(t) = \pi_i(0)e^{Qt}, \tag{9}$$

where (8) is the special form of (3) for infinite buffer model and (9) is the solution of (5).

First order, finite buffer, homogeneous behaviour. The presence of an upper boundary at B does not change the transient description a lot. It leaves the behaviour at the lower boundary, (7), unchanged, it reduces the validity of (6) to $0 < x < B$ and it introduces a differential equation, very similar to (7) for the upper boundary. That is, $u_i(t) = 0$ if $r_i < 0$ and

$$\frac{d}{dt}u_i(t) \ = r_i \ p_i(t,B) + \sum_{k\in\mathcal{S}} q_{ki} \ u_k(t), \tag{10}$$

if $r_i \geq 0$. (10) is obtained in the same way as (7).

Second order, infinite buffer, homogeneous behaviour. The case of second order Markov fluid model can be analyzed using the same method based on the short term behaviour of the Markov model. We derive the fluid density at time $t + \Delta$ based on the fluid density at time t:

- If there is no state transition in the $(t, t + \Delta)$ interval we need to evaluate a convolution with respect to the pdf of the normal distributed amount of fluid accumulated over the $(t, t+\Delta)$ interval, $f_{\mathcal{N}(\Delta r_i, \Delta\sigma_i^2)}(u)$. For simplicity we set the limits of this integration to $-\infty$ and ∞. It is to avoid the introduction of additional vanishing error terms.
- The analysis of the case with one state transition is also simplified. A convolution with finally vanishing terms should be taken into consideration in a more detailed analysis. The analysis of the term without state transition indicates how the term with one state transition vanishes, but we do not detail this point here.

$$p_i(t+\Delta,x) = (1+q_{ii}\Delta) \underbrace{\int_{-\infty}^{\infty} p_i(t,x-u) f_{\mathcal{N}(\Delta r_i,\Delta\sigma_i^2)}(u)du}_{**} +$$

$$\sum_{k\in\mathcal{S},k\neq i} q_{ki}\Delta \ \ p_k(t,x-\mathcal{O}(\Delta)) +$$

$$\sigma(\Delta)$$

To obtain the under braced term we use the Taylor expansion again, but now with 3 terms:

$$p_i(t, x-u) = p_i(t,x) - up_i'(t,x) + \frac{u^2}{2}p_i''(t,x) + \mathcal{O}(u)^3.$$

Based on this expansion we have:

$$** = p_i(t,x)\underbrace{\int_{-\infty}^{\infty} f_{\mathcal{N}(\Delta r_i, \Delta\sigma_i^2)}(u)du}_{1} - p_i'(t,x)\underbrace{\int_{-\infty}^{\infty} u f_{\mathcal{N}(\Delta r_i, \Delta\sigma_i^2)}(u)du}_{\Delta r_i} +$$
$$p_i''(t,x)\underbrace{\int_{-\infty}^{\infty} \frac{u^2}{2} f_{\mathcal{N}(\Delta r_i, \Delta\sigma_i^2)}(u)du}_{\Delta^2 r_i^2 + \Delta\sigma_i^2/2 = \Delta\sigma_i^2/2 + \sigma(\Delta)} + \underbrace{\int_{-\infty}^{\infty} \mathcal{O}(u)^3 f_{\mathcal{N}(\Delta r_i, \Delta\sigma_i^2)}(u)du}_{\mathcal{O}(\Delta)^2 = \sigma(\Delta)}.$$

Back substituting this results and performing the same steps as before we obtain

$$p_i(t+\Delta, x) = (1+q_{ii}\Delta)\left(p_i(t,x) - p_i'(t,x)\Delta r_i + p_i''(t,x)\Delta\sigma_i^2/2\right) + \sum_{k\in\mathcal{S}, k\neq i} q_{ki}\Delta\, p_k(t, x-\mathcal{O}(\Delta)) + \quad \sigma(\Delta)\ ,$$

$$p_i(t+\Delta, x) - p_i(t,x) = q_{ii}\Delta p_i(t,x) - p_i'(t,x)\Delta r_i + p_i''(t,x)\Delta\sigma_i^2/2 + \sum_{k\in\mathcal{S}, k\neq i} q_{ki}\Delta\, p_k(t, x-\mathcal{O}(\Delta)) + \quad \sigma(\Delta)\ ,$$

$$\boxed{\frac{\partial}{\partial t}p_i(t,x) + \frac{\partial}{\partial x}p_i(t,x)r_i - \frac{\partial^2}{\partial x^2}p_i(t,x)\frac{\sigma_i^2}{2} = \sum_{k\in\mathcal{S}} q_{ki}\ p_k(t,x).} \qquad (11)$$

(11) also justifies the name of this fluid models. In this case not only the first derivative of the fluid density, but also the second one appear in the partial differential equation describing the transient behaviour of the model.

Boundary condition with reflecting barrier. The boundary condition of second order Markov fluid models depends on the type of the boundary. In case of reflecting barriers the probability of empty buffer is zero, $\ell_i(t) = 0$ and the initial value of the fluid density can be computed based on (3) using (11) and (5).

Since the buffer is infinite buffer and $\ell_i(t) = 0$, we have

$$\int_0^{\infty} p_i(t,x)dx = \pi_i(t)\ .$$

Taking the derivatives of both side with respect to t results

$$\int_{x=0}^{\infty} \frac{\partial}{\partial t}p_i(t,x)\ \ dx \qquad = \frac{\partial}{\partial t}\pi_i(t)$$

Substituting (11) into the left and (5) into the right hand side we have

$$\int_{x=0}^{\infty} -\frac{\partial p_i(t,x)}{\partial x} r_i + \frac{\partial^2 p_i(t,x)}{\partial x^2} \frac{\sigma_i^2}{2} + \sum_{k \in \mathcal{S}} q_{ki}\ p_k(t,x) dx = \sum_{k \in \mathcal{S}} q_{ki} \pi_i(t),$$

from which we obtain the boundary condition as

$$-r_i \underbrace{\Big[p_i(t,x)\Big]_{x=0}^{\infty}}_{-p_i(t,0)} + \frac{\sigma_i^2}{2} \underbrace{\Big[p_i'(t,x)\Big]_{x=0}^{\infty}}_{-p_i'(t,0)} + \sum_{k \in \mathcal{S}} q_{ki} \underbrace{\int_{x=0}^{\infty} p_k(t,x) dx}_{\pi_i(t)} = \sum_{k \in \mathcal{S}} q_{ki} \pi_i(t)\ ,$$

$$\boxed{r_i p_i(t,0) - \frac{\sigma_i^2}{2} p_i'(t,0) = 0} \tag{12}$$

Fluid level dependent model behaviour. Up to now we considered fluid models where between the boundaries the fluid level does not effect the evolution of the system. It is not always the case in practice and the presented analytical description of fluid models allows to integrate fluid level dependence into the transient description in a simple way.

As a consequence we assumed that the transition rate of the discrete part of the process, q_{ij}, the mean and the variance of the fluid changing rate, r_i and σ_i, respectively, are independent of the current fluid level. When these quantities depend on the fluid level we have the following model behaviour.

$$\lim_{\Delta \to 0} \frac{Pr(S(t+\Delta) = j | S(t) = i, X(t))}{\Delta} = q_{ij}(X(t))\ .$$

When the first order model stays in state i during the $(t, t+\Delta)$ interval and the fluid level is between the boundaries

$$X(t+\Delta) - X(t) = r_i(X(t))\Delta + \sigma(\Delta),$$

and when the second order model does the same

$$X(t+\Delta) - X(t) = \mathcal{N}(r_i(X(t))\Delta, \sigma_i^2(X(t))\Delta) + \sigma(\Delta).$$

Formally it is easy to incorporate fluid level dependency into all previous equations by making the transition rates of the discrete part, the mean and the variance of the fluid changing rate depend on the fluid level, i.e., $q_{ij}(x)$, $r_i(x)$ and $\sigma_i(x)$, respectively. This ways, e. g., (6) becomes

$$\frac{\partial}{\partial t} p_i(t,x) + \boxed{r_i(x)} \frac{\partial}{\partial x} p_i(t,x) = \sum_{k \in \mathcal{S}} \boxed{q_{ki}(x)}\ p_k(t,x)\ ,$$

and the associated boundary equation, (7) becomes, if $r_i(0) < 0$ (and $r_i(x)$ is continuous):

$$\frac{d}{dt} \ell_i(t)\ = -\boxed{r_i(0)}\ p_i(t,0) + \sum_{k \in \mathcal{S}} \boxed{q_{ki}(0)}\ \ell_k(t).$$

General case. We summarize the results by providing the most general equation and present the ways to simplify it in case of special fluid models.

First of all we compose vector equations out of the set of scalar equations presented before. Let $p(t,x) = \{p_i(t,x)\}$, $\ell(t) = \{\ell_i(t)\}$ and $u(t) = \{u_i(t)\}$ be the row vectors of fluid densities, empty buffer probabilities and buffer full probabilities respectively, further more let $\mathbf{Q}(x) = \{q_{ij}(x)\}$, $\mathbf{R}(x) = \mathrm{Diag}\langle r_i(x)\rangle$ and $\mathbf{S}(x) = \mathrm{Diag}\langle\frac{\sigma_i^2(x)}{2}\rangle$ be the generator matrix of the discrete variable, the diagonal matrix of the mean fluid rates and the diagonal matrix of the variance parameter of the fluid process.

The most general equations are obtained with second order, finite buffer, fluid level dependent fluid models, where we do not define the boundary behaviour yet:

$$\begin{aligned}
&\frac{\partial p(t,x)}{\partial t} + \frac{\partial p(t,x)}{\partial x}\,\mathbf{R}(x) - \frac{\partial^2 p(t,x)}{\partial x^2}\,\mathbf{S}(x) = p(t,x)\,\mathbf{Q}(x)\,,\\
&p(t,0)\,\mathbf{R}(0) - p'(t,0)\,\mathbf{S}(0) = \ell(t)\,\mathbf{Q}(0)\,,\\
&-p(t,B)\,\mathbf{R}(B) + p'(t,B)\,\mathbf{S}(B) = u(t)\,\mathbf{Q}(B)\,,
\end{aligned} \tag{13}$$

These general equations simplify as follows, according to the boundary behaviour of the model:

- if $\sigma_i = 0$ and $r_i(x)$ is positive and continuous around zero then $\ell_i(t) = 0$, if $\sigma_i = 0$ and $r_i(x)$ is negative and continuous around B then $u_i(t) = 0$.
- if $\sigma_i > 0$ and the lower boundary is reflecting in state i then $\ell_i(t) = 0$, if $\sigma_i > 0$ and the upper boundary is reflecting in state i then $u_i(t) = 0$.
- if $\sigma_i > 0$ and the lower boundary is absorbing in state i then $p_i(t,0) = 0$, if $\sigma_i > 0$ and the upper boundary is absorbing in state i then $p_i(t,B) = 0$.

The special cases of this general case are:

- the first order model: green parts vanish,
- the infinite buffer model: blue equation vanishes,
- the fluid level independent model: $\mathbf{Q}(x)$, $\mathbf{R}(x)$, $\mathbf{S}(x)$ become $\mathbf{Q}$, $\mathbf{R}$, $\mathbf{S}$.

Normalizing condition. In case of transient analysis the set of differential equations is accompanied with an initial condition that defines the normalization of the model. Indeed the initial condition should fulfill the normalizing condition

$$\int_0^B p(0,x)\,dx\mathbb{1} + \ell(0)\mathbb{1} + u(0)\mathbb{1} = 1.$$

The set of differential equations we presented in this section preserves the probability, which means that if the initial condition satisfies the normalizing condition, then for all $t > 0$ the following normalizing condition holds

$$\int_0^B p(t,x)\ dx\mathbb{1} + \ell(t)\mathbb{1} + u(t)\mathbb{1} = 1.$$

5.4 Stationary Description of Fluid Models

The presented transient description of fluid models describes also the time limiting behaviour of these models, but as it is common with several other stochastic models, the direct stationary analysis is more efficient when only the stationary behaviour is of interest. The general approach to obtain the stationary description of fluid models is to make the t goes to infinity limit in the transient equations.

Two main questions has to be considered during this transition. If the transient functions tend to stationary values, and if this value is unique, i.e., independent of the initial condition in the sense that it converges to the same limit starting from any valid initial condition.

The typical behaviour of the above differential equations is that the solution either converges to a finite value or diverges, but does not exhibit strange behaviours like cyclic alternation, etc. Finite buffer models usually converge. To decide if an infinite buffer model converges to a proper stationary distribution we need the stability property.

Definition 2. *A fluid model is said to be* stable*, if for* $\forall x \in \mathbb{R}^+, \forall i \in \mathcal{S}$ *the time to empty the buffer*

$$T_i^E(x) = \min_{t>0}(X(t) = 0 | X(0) = x, S(0) = i)$$

has a finite mean (i.e., $E(T_i^E(x)) < \infty$*).*

Stable infinite buffer models usually converge. It is easy to decide if a model is stable in case of fluid level independent Markov fluid models. The condition of stability is

$$\sum_{i \in \mathcal{S}} \pi_i r_i < 0,$$

where π_i is the stationary distribution of the discrete part of the model. The stability of fluid level dependent Markov fluid models is more complex to decide. It requires the solution of the differential equations describing the stationary behaviour of the process.

To decide if the stationary behaviour is unique we need the ergodic property.

Definition 3. *A fluid model is said to be* ergodic*, if for* $\forall x, y \in \mathbb{R}^+, \forall i, j \in \mathcal{S}$ *the transition time*

$$T_{i,j}(x,y) = \min_{t>0}(X(t) = y, S(t) = j | X(0) = x, S(0) = i)$$

has a finite mean (i.e., $E(T) < \infty$*).*

The stationary behaviour of ergodic fluid models is independent of the initial condition.

Stationary equations. Assuming the following limits exists, describe a proper distribution and independent of the initial condition we present the stationary equations obtained from the transient ones.

- $\pi_i = \lim_{t\to\infty} Pr(S(t) = i)$ – state probability,
- $u_i = \lim_{t\to\infty} Pr(X(t) = B, S(t) = i)$ – buffer full probability,
- $\ell_i = \lim_{t\to\infty} Pr(X(t) = 0, S(t) = i)$ – buffer empty probability,
- $p_i(x) = \lim_{t\to\infty} \lim_{\Delta\to 0} 1/\Delta \;\; Pr(x < X(t) < x + \Delta, S(t) = i)$ – fluid density,
- $F_i(x) = \lim_{t\to\infty} Pr(X(t) < x, S(t) = i)$ – fluid distribution.

The stationary counterpart of (13) can be obtained by making the t goes to infinity limit on both sides of the equations:

$$\begin{aligned} p'(x)\,\mathbf{R}(x) - p''(x)\,\mathbf{S}(x) &= p(x)\,\mathbf{Q}(x)\,, \\ p(0)\,\mathbf{R}(0) - p'(0)\,\mathbf{S}(0) &= \ell\,\mathbf{Q}(0)\,, \\ -p(B)\,\mathbf{R}(B) + p'(B)\,\mathbf{S}(B) &= u\,\mathbf{Q}(B)\,, \end{aligned} \tag{14}$$

where the fluid rate and the boundary conditions determine the following variables:

- if $\sigma_i = 0$ and $r_i(x)$ is positive and continuous around zero then $\ell_i = 0$, if $\sigma_i = 0$ and $r_i(x)$ is negative and continuous around B then $u_i = 0$,
- if $\sigma_i > 0$ and the lower boundary is reflecting in state i then $\ell_i = 0$, if $\sigma_i > 0$ and the upper boundary is reflecting in state i then $u_i = 0$,
- if $\sigma_i > 0$ and the lower boundary is absorbing in state i then $p_i(0) = 0$, if $\sigma_i > 0$ and the upper boundary is absorbing in state i then $p_i(B) = 0$.

Normalizing condition. In case when the stationary solution is computed based on (14), we cannot utilize the information about the initial condition of the model, but the solution must fulfill the normalizing condition:

$$\int_0^B p(x)\; dx\mathbb{1} + \ell\mathbb{1} + u\mathbb{1} = 1.$$

6 Solution Methods

There are several different ways to evaluate Markov fluid models. They differ in their applied analysis approach, provided results and applicability. It is possible to obtain symbolic solution for rather small models (Markov fluid models with less than 5 discrete states), but for larger models the application of numerical methods is feasible only. Table 1 presents a summary of some potential

Table 1. Solution methods for Markov fluid models

	transient	statioanry
differential equations	[8]	[16]
spectral decomposition	+	[23,1,7,6]
randomization	[34]	[36,35]
transform domain	[33]	+
Markov regenerative	[2]	+
matrix exponent	+	[18]

approaches and classifies some research papers according to their applied approaches. In this section we summarize some analysis approaches, but we do not intend to provide a complete view.

Table 1 indicates that all of the mentioned solution methods are applicable to both, the transient and the stationary analysis, but in a different way. In case of transient analysis we have a set of partial differential equations (13), a set of boundary conditions, and a set of explicit initial conditions. Starting from this initial condition it is possible to evaluate the model behaviour using a forward analysis approach. In case of stationary analysis we have a set of ordinary differential equations (14), a set of boundary conditions, and a normalizing conditions. A difficulty of the stationary analysis with respect to the transient one is that normalizing condition does not provide an explicit expression to start the solution from. Apart of this the transient analysis is more complex than the stationary one, since we have one variable (t) more in the transient case.

In the rest of this section we summarize the main ideas of some selected solution methods.

6.1 Transient Solution Methods

Numerical solution of differential equations. Chen et al. proposed a discretization based numerical technique to evaluate the transient behaviour of fluid models [8]. The main strength of their approach is that that all mentioned model behaviour can be analyzed with it. Indeed this is the only approach for the transient analysis of fluid level dependent models. The proposed approach starts from the initial condition, and computes (approximates) the evolution of the fluid distribution step-by-step in Δ long time intervals at some fluid levels based on the differential equations and the boundary condition.

Randomization. Randomization is an effective numerical analysis approach that is widely used for the transient analysis of CTMCs, i.e., for the numerical solution of (5). It is numerically stable procedure where the convex combination of probabilities (non-negative numbers less or equal to one) are computed. The procedure is based on a symbolic solution of (5). Sericola extended this technique to the transient analysis of first order, infinite buffer, fluid level independent Markov fluid models [34]. Indeed he provided a symbolic solution of (6) in the following form:

$$F_i^c(t,x) = \sum_{n=0}^{\infty} e^{-\lambda t} \frac{(\lambda t)^n}{n!} \sum_{k=0}^{n} \left\langle\!\!\left\langle {n \atop k} \right\rangle\!\!\right\rangle x_j^k (1-x_j)^{n-k} b_i^{(j)}(n,k),$$

where $F_i^c(t,x) = Pr(X(t) > x, S(t) = i)$, $x_j = \frac{x - r_{j-1}^+ t}{r_j t - r_{j-1}^+ t}$ if $x \in [r_{j-1}^+ t, r_j t)$, and $b_i^{(j)}(n,k)$ is defined by initial value and a simple recursion.

The main properties of this randomization based solution method are as follows:

- the expression with the given recursive formulas is a solution of the differential equation,
- the initial value of $b_i^{(j)}(n,k)$ is set to fulfill the boundary condition,
- due to the fact that $0 \leq x_j \leq 1$ we have the same numerical stability properties as for the transient analysis of CTMCs:
  convex combination of non-negative numbers are computed, and hence the floating point errors has a limited effect and it does not cause problems like "ringing" (change of sign),
- the initial fluid level must be $X(0) = 0$ (extension to $X(0) > 0$ and to finite buffer is not available).

Markov regenerative approach. Ahn and Ramaswami recommended to divide the transient analysis of first order, infinite buffer, fluid level independent Markov fluid models into periods according to the busy/idle state of the buffer [2]. When T_i is the beginning of the ith busy (non-empty) period of the fluid buffer then the $(S(t_i), T_i)$ pairs form a Markov renewal sequence. The analysis of a busy and an idle cycle, i.e., a (T_{i-1}, T_i) interval, is divided into two parts. The idle period is easier to analyze. Its length is phase type distributer. The analysis of the busy period is more complex, but Ahn and Ramaswami recognized the similarities between fluid and queueing models and provided a solution method based on Matrix analytic technique.

Transform domain description. Ren and Kobayashi proposed a solution technique based on the Laplace transform domain description first order, infinite buffer, fluid level independent Markov fluid models [33]. The Laplace transform of (6) is

$$p^{**}(s,v) = (\underbrace{p^*(0,v)}_{\text{initial condition}} + \underbrace{p^*(s,0)}_{\text{unknown}} \mathbf{R})(s\mathbf{I} + v\mathbf{R} - \mathbf{Q})^{-1}.$$

where $p^{**}(s,v)$ must be analytical. Since $p^*(0,v)$ is known from the initial condition $p^*(s,0)$ is set to eliminates the roots of $\det(s\mathbf{I} + v\mathbf{R} - \mathbf{Q})$.

This approach provides a closed form solution also for the case of initially non-empty buffer ($X(0) > 0$), but its applicability is limited to small models (less than 5 discrete states) since it is based on complex symbolic functional analysis.

6.2 Stationary Solution Methods

Spectral decomposition. One of the first papers on the application of Markov fluid models for modeling of telecommunication systems [7] already applied the spectral decomposition method for the solution of the obtained model. Later on Kulkarni presented a survey on spectral decomposition based analysis of first order, infinite and finite buffer, fluid level independent Markov fluid models [23].

To present these results we need the following notations. The set of discrete states are partitioned as follows:

- $\mathcal{S}^+$: $i \in \mathcal{S}^+$ iff $\sigma_i > 0$ – second order states,
- $\mathcal{S}^0$: $i \in \mathcal{S}^0$ iff $r_i = 0$ and $\sigma_i = 0$, – zero states,
- $\mathcal{S}^{0+}$: $i \in \mathcal{S}^{0+}$ iff $r_i > 0$ and $\sigma_i = 0$, – positive first order states,
- $\mathcal{S}^{0-}$: $i \in \mathcal{S}^{0-}$ iff $r_i < 0$ and $\sigma_i = 0$, – negative first order states,
- $\mathcal{S}^* = \mathcal{S}^{0-} \bigcup \mathcal{S}^{0+}$, – first order states.

The general form of the solution of the differential equation $p'(x)\mathbf{R} = p(x)\mathbf{Q}$ is

$$p(x) = e^{\lambda x}\phi,$$

where ϕ is a row vector. Substituting this solution into the differential equation we get the characteristic equation:

$$\phi(\lambda\mathbf{R} - \mathbf{Q}) = 0,$$

whose solutions are obtained at

$$\det(\lambda\mathbf{R} - \mathbf{Q}) = 0.$$

The characteristic equation has $|\mathcal{S}^{0+}| + |\mathcal{S}^{0-}|$ solutions, with $|\mathcal{S}^{0+}|$ negative eigenvalues, 1 zero eigenvalue, and $|\mathcal{S}^{0-}| - 1$ positive eigenvalues. Having these eigenvalues and eigenvectors the solution is

$$p(x) = \sum_{j=1}^{|\mathcal{S}^{0+}|+|\mathcal{S}^{0-}|} a_j e^{\lambda_j x}\phi_j,$$

and the a_j coefficients are set to fulfill the boundary and normalizing conditions. In the *infinite buffer* case these conditions are:

- $p(0)\ \mathbf{R}\ = \ell\ \mathbf{Q}$,
- $\ell_i = 0$ if $r_i > 0$, and
- $\int_0^\infty p_i(x)dx + \ell_i = \pi_i$.

From which $a_j = 0$ for $\lambda_j > 0$ and the a_j coefficients for $\lambda_j < 0$ are obtained from the solution of the linear system of equations determined by the conditions of infinite buffer.
In the *finite buffer* case these conditions are:

- $p(0)\ \mathbf{R}\ = \ell\ \mathbf{Q}$, $p(B)\ \mathbf{R}\ = u\ \mathbf{Q}$,

– $\ell_i = 0$ if $r_i > 0$, $u_i = 0$ if $r_i < 0$, and
– $\int_0^\infty p_i(x)dx + \ell_i + u_i = \pi_i$.

From which all a_j coefficients are obtained from the linear system of equations determined by the conditions of infinite buffer.
The result on the sign of the eigenvalues has the following consequences:

– If $|\mathcal{S}^{0-}| > 1$ and the buffer is infinite then there is at least one positive eigenvalue, which needs to be excluded from the solution (if the fluid model is stable). The exclusion of the positive eigenvalue makes the spectral decomposition necessary.
– If $|\mathcal{S}^{0-}| = 1$ and the buffer is infinite, then all eigenvalues are non-positive and there is no need to exclude any eigenvalue from the solution.
– If the buffer is finite all eigenvalues plays role in the solution, i.e., there is no need for special treatment of the positive eigenvalues.

Matrix exponent. An algebraic approach was proposed by Gribaudo and German to solve the set of equations given for first order, finite buffer, fluid level independent Markov fluid models [18]. Assuming that $|\mathcal{S}^0| = 0$ and $\mathcal{S} = \mathcal{S}^*$ they introduced $v = \ell + u$, $\mathbf{Q}^-$, $\mathbf{Q}^+$, where $q^-_{ij} = q_{ij}$ if $i \in \mathcal{S}^-$ and otherwise $q^-_{ij} = 0$. With these notations the set of equations becomes:

$$\begin{aligned}
\frac{\partial p(x)}{\partial x}\mathbf{R} = p(x)\mathbf{Q} \quad &\longrightarrow \quad p(B) = p(0)\ e^{\mathbf{Q}\mathbf{R}^{-1}B} = p(0)\ \mathbf{\Phi}, \\
p(0)\mathbf{R} = v\mathbf{Q}^- \quad &\longrightarrow \quad p(0) = v\mathbf{Q}^-\mathbf{R}^{-1}, \\
-p(B)\mathbf{R} = v\mathbf{Q}^+ \quad &\longrightarrow \quad \boxed{v(\mathbf{Q}^-\mathbf{R}^{-1}\mathbf{\Phi}\mathbf{R} + \mathbf{Q}^+) = 0},
\end{aligned}$$

where the equation in the box is linear for the unknown element of vector v. The normalizing condition of this equation is

$$\ell\mathbb{1} + u\mathbb{1} + p(0)\underbrace{\int_0^B e^{\mathbf{Q}\mathbf{R}^{-1}x}dx}_{\mathbf{\Psi}}\mathbb{1} = \boxed{v(\mathbf{I} + \mathbf{Q}^-\mathbf{R}^{-1}\mathbf{\Psi})\mathbb{1} = 1}.$$

Relation of spectral decomposition and matrix exponent. With some rearrangement of the spectral solution we can show that the above presented 2 solutions are identical. Suppose that $|\mathcal{S}^0| = 0$ and $\mathcal{S} = \mathcal{S}^*$ the characteristic equation is $\phi(\lambda\mathbf{I} - \mathbf{Q}\mathbf{R}^{-1}) = 0$, and the spectral solution is $p(x) = \sum_{j=1}^{|\mathcal{S}|} a_j e^{\lambda_j x}\phi_j$, where λ_j and ϕ_j are the eigenvalues and the left eigenvector of matrix $\mathbf{Q}\mathbf{R}^{-1}$. Introducing vector $a = \{a_j\}$ and matrix $\mathbf{B} = \begin{pmatrix} \phi_1 \\ \hline \phi_2 \\ \hline \vdots \\ \hline \phi_{|\mathcal{S}^*|} \end{pmatrix}$, the spectral solution can be rewritten as

$$p(x) = \sum_{j=1}^{|\mathcal{S}|} a_j e^{\lambda_j x} \phi_j = a \ \mathrm{Diag}\langle e^{\lambda_i x}\rangle \ \mathbf{B}$$
$$= \underbrace{a \ \mathbf{B}}_{p(0)} \ \underbrace{\mathbf{B}^{-1} \ \mathrm{Diag}\langle e^{\lambda_i x}\rangle \ \mathbf{B}}_{e^{\mathbf{Q}\mathbf{R}^{-1}x}},$$

which is the matrix exponential form used in [18].

Spectral decomposition of second order models. The spectral decomposition based analysis of second order, infinite and finite buffer, fluid level independent Markov fluid models is presented by Karandikar and Kulkarni in [22]. In this case the differential equation has the form $p'(x)\ \mathbf{R}\ -\ p''(x)\ \mathbf{S}\ = p(x)\ \mathbf{Q}$. The general form of the solution if this equation is the same as in the first order case, $p(x) = e^{\lambda x}\phi$, but back substituting this solution we a different characteristic equation:

$$\phi(\lambda \mathbf{R} - \lambda^2 \mathbf{S} - \mathbf{Q}) = 0.$$

This characteristic equation has $2|\mathcal{S}^+|+|\mathcal{S}^*|$ solutions, with $|\mathcal{S}^+|+|\mathcal{S}^{0+}|$ negative eigenvalues, 1 zero eigenvalue, and $|\mathcal{S}^+|+|\mathcal{S}^{0-}|-1$ positive eigenvalues. The final form of the solution is

$$p(x) = \sum_{j=1}^{2|\mathcal{S}^+|+|\mathcal{S}^*|} a_j e^{\lambda_j x} \phi_j,$$

and the a_j coefficients are set to fulfill the boundary and normalizing conditions.

A transformation of the quadratic equation to a linear one. To avoid handling quadratic equations several authors recommended to transform the system into a linear one with enlarged size, e.g., [6]. In case of second order, infinite and infinite buffer, fluid level independent models with $|\mathcal{S}^0| = |\mathcal{S}^*| = 0$ and $\mathcal{S} = \mathcal{S}^+$, this transformation is based on the following representation of the differential equation

$$\frac{d}{dx}p(x)\ \mathbf{R}\ -\ \frac{d}{dx}p'(x)\ \mathbf{S}\ = p(x)\ \mathbf{Q}\ ,$$
$$\frac{d}{dx}p(x)\ \mathbf{I}\ = p'(x)\ \mathbf{I}\ .$$

This equations form a vector equation of double size

$$\frac{d}{dx}\boxed{p(x)}\boxed{p'(x)}\ \begin{array}{|c|c|}\hline \mathbf{R} & \mathbf{I} \\ \hline -\mathbf{S} & \mathbf{0} \\ \hline \end{array} = \boxed{p(x)}\boxed{p'(x)}\ \begin{array}{|c|c|}\hline \mathbf{Q} & \mathbf{0} \\ \hline \mathbf{0} & \mathbf{I} \\ \hline \end{array}.$$

Introducing $\hat{p}(x) = \boxed{p(x)}\boxed{p'(x)}$, $\hat{\mathbf{R}} = \begin{array}{|c|c|}\hline \mathbf{R} & \mathbf{I} \\ \hline -\mathbf{S} & \mathbf{0} \\ \hline \end{array}$, and $\hat{\mathbf{Q}} = \begin{array}{|c|c|}\hline \mathbf{Q} & \mathbf{0} \\ \hline \mathbf{0} & \mathbf{I} \\ \hline \end{array}$, we obtain a first order differential equation

$$\frac{d}{dx}\hat{p}(x)\ \hat{\mathbf{R}}\ = \hat{p}(x)\ \hat{\mathbf{Q}}\ \ ,$$

whose solution is

$$\hat{p}(B) = \hat{p}(0)\ e^{\hat{\mathbf{Q}}\hat{\mathbf{R}}^{-1}B}.$$

Randomization. One of the simplest stationary solution methods is based on randomization. It is applicable for first order, infinite [36] or finite [35] buffer, fluid level independent Markov fluid models. A symbolic solution of the differential equation is

$$F_i(x) = \sum_{n=0}^{\infty} e^{-\lambda t/r} \frac{(\lambda t/r)^n}{n!}\ b_i(n)$$

where $r = \min(r_i | r_i > 0)$ and $b_i(n)$ is defined by simple recursion such that the boundary conditions are fulfilled. Similar to other randomization based methods the numerical procedure computes convex combination of non-negative numbers, which ensures nice numerical properties.

The main limitation of these randomization based methods is that $|\mathcal{S}^{0-}|$ must be 1. Extension to the $|\mathcal{S}^{0-}| > 1$ case is not known.

Numerical solution of differential equations. Unfortunately non of the above stationary analysis methods is applicable with fluid level dependent models. The only approach that is applicable for fluid level dependent cases is based on the numerical solution of the

$$\mathbf{M}'(x)\,\mathbf{R}(x) - \mathbf{M}''(x)\,\mathbf{S}(x) = \mathbf{M}(x)\,\mathbf{Q}(x) \tag{15}$$

differential equation with initial condition $\mathbf{M}(0) = \mathbf{I}$, as it is proposed by Gribaudo et al. in [16].

The solution is composed by the following steps:

- Numerically solve the matrix function $\mathbf{M}(x)$ based on the differential equation (15)
- calculate the unknowns $(p(0), p(B), \ell, u)$ based on the boundary conditions, the normalizing condition and

$$p(B) = p(0)\ \mathbf{M}(B)$$

The major limitation of this approach is that it limited to finite buffer models.

7 Application

Fluid Models and FSPNs have been successfully used in the literature to study several interesting systems. Here we present how Fluid Stochastic Petri Nets have been used in [27] to compute the transfer time distribution of resource in a Peer-to-Peer file sharing application.

File transfer using Peer-to-Peer file sharing applications is usually divided into two steps: resource search and resource download. Depending on the file

size and its popularity, either of the two phases can become the bottleneck. In this section we describe both the location and download phases of a generic Peer-to-Peer file sharing application using a fluid model. We propose a model that allows the computation of the transfer time distribution, and that it is capable of considering some advanced characteristic such as parallel downloads and on-off peer behavior. These features, although quite common in the real applications, have not been considered in previous models proposed in the literature. Model parameters reflect network, application, resource and user characteristics, and can be tuned to analyze a large number of different real implementations.

Peer-to-Peer Model. The proposed fluid model for the estimation of the transfer time distribution in P2P file sharing applications will be described using the Fluid Stochastic Petri Net (FSPN) formalism [21,20]. Table 2 reports the other notations derived from the reference.

Table 2. Model Notations

Notation	Description	Range
$\mathcal{B}$	Set of bandwidths	{14.4, 28.8, 33.6, 56, 64 128, DSL, Cable, T1, T3}
N	Number of peers holding the resource	$\mathbb{N}$
SB	Server bandwidth	$\mathcal{B}$
CB	Tagged client bandwidth	$\mathcal{B}$
S	Resource size	$\mathbb{N}$
$K(b)$	Max. number of concurrent peers	$\mathcal{B} \rightarrow \mathbb{N}$
LT	Average number of requests of uploads	$\mathbb{N}$
W	Bandwidth dependent weight	$\mathcal{B} \rightarrow [0,1]$
$L(b)$	LT as function of the peer bandwidth	$LT * W(b)$

The FSPN basic Model The basic model [14] computes the transfer time distribution of a resource of size S downloaded by a client with a bandwidth CB, from a server with bandwidth SB. It neglects both the search and the queueing phase, and download interruptions. That model is defined by means of a FSPN. The main assumption in the basic model, is that the session time of concurrent peers is described by an Hyperexponetnial distribution (with parameters α, μ_1 and μ_2), and that the interarrival time of concurrent downloaders is approximated by an exponential distribution (whose parameter $L(cb)$ is bandwidth dependent). The maximum number of concurrent downloads from a server is limited by a bandwidth dependent parameter $K(sb)$. Moreover, the server bandwidth is equally shared among the concurrent downloaders. For a discussion on the validity of these assumptions, please refer to [14].

Using these assumptions, the available bandwidth at the client can be computed as a function of the number of concurrent peers. In particular, if we call I_j the total number of concurrent peers in a discrete state of FSPN model, then the available bandwidth is equal to:

$$f(I_j) = \min\left(\frac{sb}{I_j + 1}, cb\right) \quad . \tag{16}$$

The FSPN model is analyzed by solving the system of partial derivative differential equations that describes its underlaying stochastic process. From the

solution to these equations the probability density $\bar{\pi}(\tau, x)$ of the fluid level at a given time instant τ can be directly computed. $\bar{\pi}(\tau, x)$ corresponds to the probability density that the number of bytes downloaded at time τ is equal to x. By integrating this quantity, the probability distribution that a file of size s can be downloaded in less than t can be computed:

$$F_t(t|s) = \int_s^\infty \bar{\pi}(\tau, x)dx \Big|_{\tau=t}. \tag{17}$$

Modeling the search time, queueing time and peer unavailability. Search time is conditioned by many factors such as the popularity of the resource, protocol characteristics, the participation level of the user and the number of neighbor peers. After the searching phase the client selects peers from which get the resource. Queueing time is the time spent before a selected server serves the client request. It also depends on many factors, as the number of concurrent downloads allowed, the bandwidth of the server and the number of concurrent clients, the protocol, and the participation level of clients.

Creating a detailed model to consider all these aspects would be too complex. Instead we simplify the model by considering *the aggregate search plus queuing time perceived by a client*. That is, we suppose that we could compute the distribution $QS(\tau)$ of the time required from the start of the search to the start of the actual download of a resource. This seems to be a quite strong assumption, but we will prove, at the end of this section, that despite its simplicity, the proposed model is able to get most of the qualitative features that characterize parallel download in peer to peer applications.

Figure 13 represents the extension of the model proposed [14]. The arrival of a new concurrent download is modelled by transition request_arrival. The session length distribution is modelled by the sub-net composed by places CHOICE, STAGE_1, STAGE_2, END_SERVICE and transitions choose_1, choose_2, terminate_service, service_1, service_2. Their parameters are directly mapped to the parameters of the distributions outlined in Section 7. The maximum number of concurrent downloads is determined by the initial marking of place AVAILABLE, and is set according to parameter $K(sb)$. The amount of byte transferred is modelled by fluid place TRANSFERRED and fluid transition transfer. The value of parameter I_j corresponds to the sum of the marking of places STAGE_1 and STAGE_2. The search and queueing phases are represented by the generic firing time transition TON, with distribution ϕ_{on}.

Due to the active/non-active peer dynamics the server may become unavailable and then its service is stopped. When failures occur, the client starts a new search of the same resource, and then it continues the download (likely from another peer), after experiencing a new queueing time. The failure of server is represented in the model by generic firing time transition TOFF, with firing time distribution ϕ_{off}.

Place SandQ represents the search and queueing phases, and place TRANS the resource transfer phase.

As reported in [14], special care should be used to compute the initial distribution of the number of concurrent peers at the server. In this case, the initial state of the places representing the concurrent peers at the server, should be determined at the time when the actual transfer starts, i.e. at the firing of transition TON. When transition TON fires, it should set the number of tokens in places AVAILABLE, STAGE_1 and STAGE_2 according to the initial distribution, determined following the technique proposed in [14]. The setting of the initial state is achieved by an appropriate set of immediate transitions, weighted according to the initial state distribution. In order to simplify Figure 13, this sub-net has been removed and has been represented by the gray arrow labeled with *Set Initial state*. Similarly, when the server experience a failure, all the places of the sub-model representing its state must be emptied. This also can be achieved by an appropriate set of immediate transition, which has been represented in Figure 13 by the gray arrow labeled with *Clear state*.

In this model, the popularity of the resource is considered when determining the rate of transition TON. A very popular resource will have a shorter search and queueing time, since will be available from more peers. A rare resource will instead have a very high searching and queueing time.

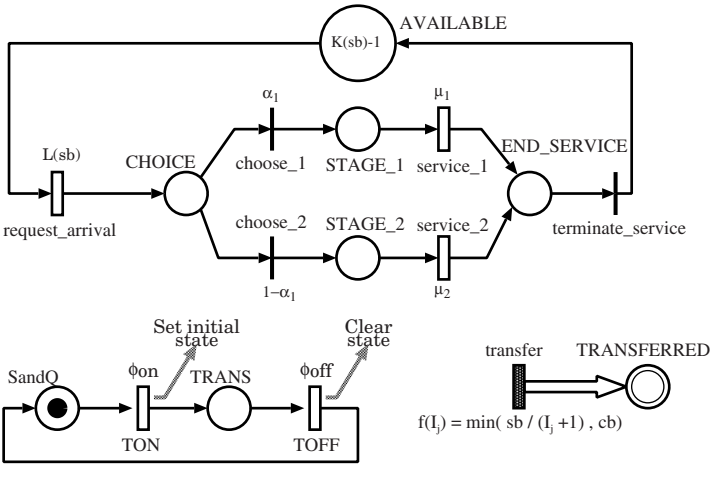


Fig. 13. FSPN model representation of an unreliable server with search and queueing phases

Considering the parallel download from multiple sources. The model that represents parallel download from multiple servers can be obtained by repeating H times the sub-models of Figure 13 representing the server and the search-queueing state, where H corresponds to the maximum number of parallel downloads. This is represented in Figure 14. Note that the H sub-models representing the H servers, share the same resource download buffer, modeled by fluid place TRANSFERRED. In this case, the rate at which the file is downloaded,

Table 3. Model parameters used for experiments

Service parameters	
μ_1	0.001
μ_2	0.1
α_1	0.6
α_2	0.4
Arrival rate	
LT	0.01

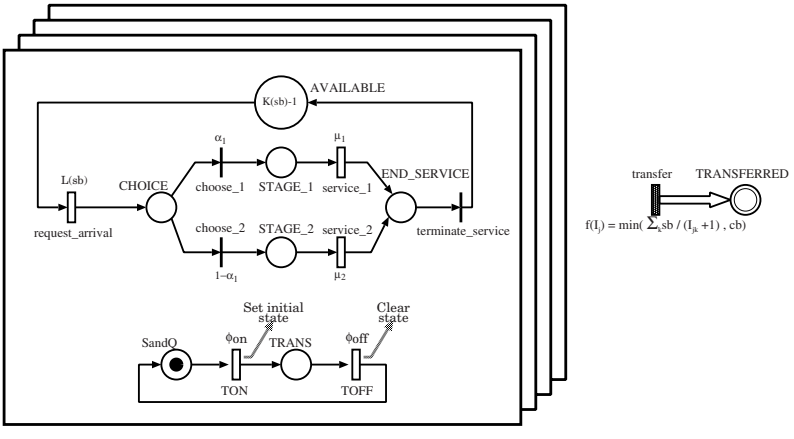


Fig. 14. FSPN model for multiple servers download

is expressed as the minimum between the client bandwidth cb, and the sum of the download rate from each server that is active in that time instant, that is:

$$f(I_j) = \min\left(\sum_{k=1}^{H} \mathcal{I}(\#\text{TRANS}_k = 1)\frac{sb_k}{I_{jk}+1}, cb\right) \tag{18}$$

where $\mathcal{I}(\#\text{TRANS}_k)$ is an indicator function that returns 1 if the the number of tokens in place TRANS of the submodel representing the k-th server is equal to 1 (i.e. active download), zero otherwise. $I_{jk}+1$ represents the sum of the tokens in places STAGE_1 and STAGE_2 for each tangible (discrete) state $\boldsymbol{m}_j$ of the k-th server, i.e. the number of requests that interfere on that server with the tagged client service.

Despite the symmetries, the sub-models are not independent, since they are coupled by the fluid buffer TRANSFERRED. Moreover the relation that governs the rate of the growth of the fluid place (Equation 18) is non-linear, due to the presence of the min(·) function. This prevents to apply a solution technique that analyzes each server separately, and combine them afterward.

Table 4. Downloading bandwidth versus session duration and resource size

Resource Size	Average Bandwidth (Kbit/sec)		
	Session Time 1000 sec	Session Time 20 sec.	Session Time 10 sec.
512 KB	24.32	10.4	6.56
4 MB	59.68	14.08	7.84
10 MB	68.48	6.64	3.12

Experiments. The proposed models, despite their simplifying assumptions, can describe the qualitative behavior of real peer to peer systems. In both cases, we present our analysis only for the case when all peers have the same bandwidth connection (in particular, we consider 640 MB/s DSL technology). We also approximate both search and queue time distribution and the server failure distribution with exponential distributions. For this reason, in the following we will use parameters ϕ_{on} and ϕ_{off} to indicate the rate of the corresponding exponential distribution.

Extensive validation of results for our modeling technique shares the same difficulty of previous works on analytical models for P2P systems. It is a difficult task since existing measurement based studies have not focused on characterizing the duration of the transfer phase. Although it might be possible to validate our model through detailed simulations of realistic P2P file sharing applications it would have a prohibitive programming and computational cost. Nevertheless, we performed simple validations by comparing model results in selected cases where theoretical results are known or can be exactly computed. In particular, we compared model results with the ideal case where there is no competition for the server bandwidth and the transfer is only conditioned by the minimum bandwidth between server and clients. In these cases we found a perfect agreement between the model predictions and the theoretical results. It is a safety check that allow us to know that at least in the deterministic case, without concurrent operations, model result is identical to the expected one (that is the ratio between the resource size and the minimum bandwidth among client and server ones). Moreover, results presented in Table 5 are partially supported by the measurement study presented in [30]. In particular, in [30] it is shown that the average download speed is $30KB/sec$ that in the case of a $4MB$ resource corresponds to an average transfer time of $133seconds$. This average is comparable with most of average values, referring to different number of sources, shown in Table 5.

A first intuitive result (see Table 4) shows that the transfer time increases with the increasing of unavailability rate. However, we must point out that this effect heavily depends on the resource size. We thus perform an analysis with respect to the resource size, in particular, we look at average bandwidth experienced during the file transfer as function of the failure rate. We keep the searching-queueing rate constant to 0.01: this means that client wait a mean of 100 seconds to find a new connection. We vary the failure rate in order to get server sessions of 10, 20, and 1000 seconds. The number of concurrent peers on the server, $K(sb)$-1

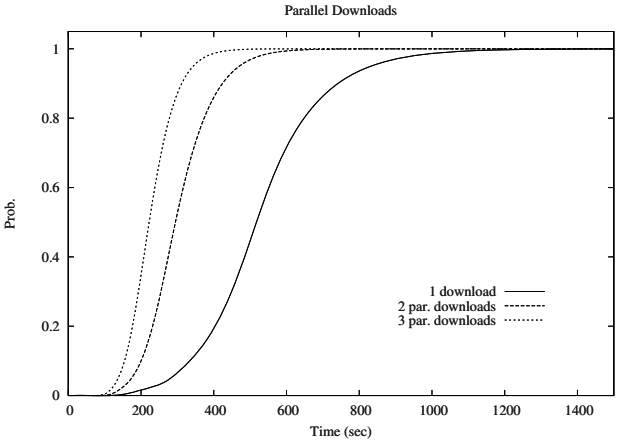


Fig. 15. Improvement provided by parallel downloads

Table 5. Transfer time as function of the number of sources

Number of Sources	Tansfer Time (sec.)		
	Mean	50^{th} quantile	90^{th} quantile
3	178	170	240
4	148	140	200
5	131	130	170
6	119	120	160
7	110	110	148
8	104	100	130
9	104	100	130

(minus one takes into account the tagged client), is set to 3. In this analysis we does not consider parallel downloads. FSPN model parameters used in this experiment are reported in Table 3 while results are shown in Table 4. The index we use to evaluate the performance is the average bandwidth experienced to complete the transfer of the resource. It has been computed as the ratio between the resource size and the average of the time transfer. It is interesting to note that bigger resources suffer significantly from servers failure. For instance, in the case of a 10 MBytes resource, the bandwidth falls down when the failure rate is 0.05 and 0.1 (that is session time of 10 and 20 seconds). Instead in the case of a 512 KByte resource, the penalty introduced by the failure of the server is less significative. This is due to fact that, on the average, the resource can be completely transferred before the server fails, despite shorter server session.

Most P2P file sharing applications (e.g., eDonkey, BitTorrent, etc.) allow parallel downloads. The model presented in Figure 14 represents this feature. Client peer downloads from multiple sources and gets better performance when the number of source increases as shown in Fig. 15. This experiment refers to the transfer of a 4 MByte file, with searching-queueing rate equal to 0.01 and failure rate equal to 0.001, the number of concurrent peers on each server, $K(sb) - 1$

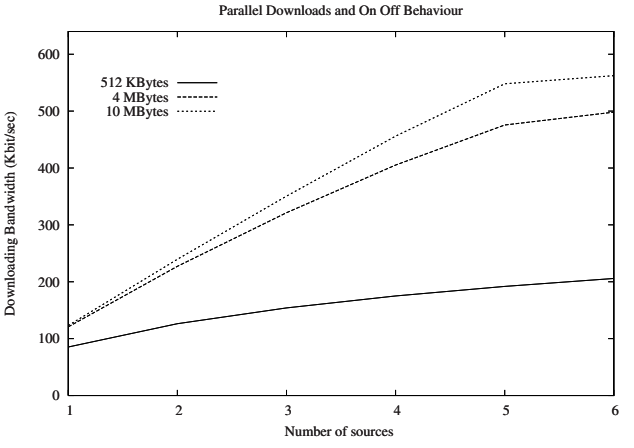


Fig. 16. Benefits of parallel downloading for different resource sizes

Table 6. Modeling searching and queueing phases with different distributions

Distribution	Tansfer Time (sec.)		
	Mean	90^{th} quantile	95^{th} quantile
Hypo-exponential	448	580	630
Exponential	376.31	575	660
Hyper-exponential 1	306.34	495	605
Hyper-exponential 2	297.35	440	510

(minus one takes into account the tagged client), is set to 3. However, improvements in performance are limited by the client download bandwidth; i.e. when the total bandwidth provided by multiple servers exceeds the maximum client download bandwidth, the speed at which the file is transferred remains constant, despite the growth in the number of sources. This is shown in table 5, where the mean and quantiles of the transfer time distribution related to a 4 MBytes resource are reported as function of the number of sources. In this case parameters are: searching-queueing rate equal to 0.01 and failure rate equal to 0.001, the number of concurrent peers on each server, $K(sb) - 1$, is set to 1. We can note that the improvement in transfer performance become less significative as the number of sources increases (since they saturate the client downloading bandwidth). When sources become 9 the time required to transfer the file remains constant. This insight may provide suggestions for the application design. E.g., let suppose that the application protocol is able to monitor the client bandwidth status. If it detects that the client is the bottleneck, then it can avoid to add new (parallel) sources. Their contribute, that should not be exploited in order to improve tagged transfer performance, could be exploited to improve the system service capacity for other peers.

It is interesting to see how the benefit derived from the use of parallel download depends on the size of the resource. Consider the case in which downloading

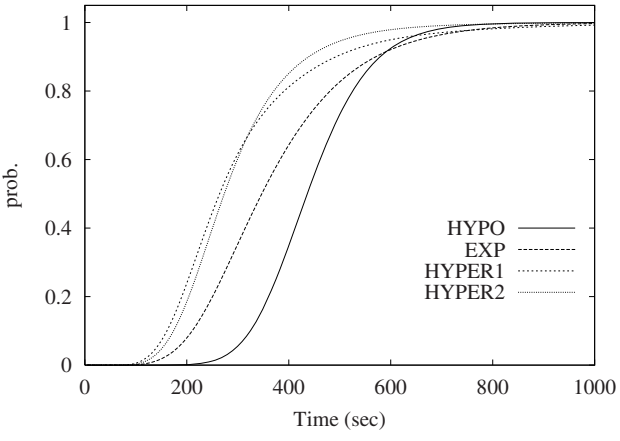


Fig. 17. Transfer time distribution with different searching-queueing rate distributions

session does not suffer from the servers failures (i.e. the failure rate is very low). We set searching-queueing rate much bigger than the failure one, respectively 0.1 and 0.001. The number of concurrent peers on each server, $K(sb) - 1$, is set to 1. The study has been done for 512 KBytes, 4 and 10 MBytes resource sizes and for a number of parallel downloads that grows from 1 up to 6, as shown In Fig. 16. Small resources take less benefits from parallel downloading, since the downloading time is shorter than the time required by the searching and queueing phase to start a parallel download from another source. For bigger resources instead, the downloading time is reduced significantly with the increases in the number possible download source. These improvements are however limited by the client bandwidth, as shown in the previous example. This can be seen for for the 4 and the 10 MBytes cases, when the number of sources increase from 5 to 6.

In order to describe different system scenario we also approximate the searching and queueing rate with different distributions. All previous results refer to the exponential case. In addition we model the searching and queueing phases with Hyper-exponential and Hypo-exponential distributions. Results are reported in Table 6; Figure 17 refers to the transfer of a $4MB$ file with 3 parallel downloads and a session mean time of 15 minutes. In all cases the mean time spent in the searching/queueing phase is 5 minutes. In the case "Hyper-exponential 1" the mean time spent by the client is 10 minutes with a probability of 44% and 1 minute with a probability of 56%. In this case faster searching/queueing phases are favorite, indeed transfer time is shorter than in the case "Exponential". Shorter searching/queueing phases are even more favorite in the case "Hyper-exponential 2": 3.45 minutes with probability 80%, and 10 minutes with probability 20%. This setting results in faster transfers, as reported in Table 6. The choice of the Hyper-exponential distribution can be useful for describing different scenario where shorter searching/queueing phases model popular resource

transfers and longer ones model rare resource transfers. The Hypo-exponential distribution can be used to model rates when the approximation should be more deterministic. In this case, the Hypo-exponential case corresponds to a 5 stages Erlang distribution. Even if the goal of this work is not to compare different approximations, it shows that the proposed model can be considered a flexible tool for evaluating P2P applications performance.

8 Conclusions

Stochastic models with continuous variables (Reward models, Fluid models and FSPNs) often allows proper modeling of real systems. Their analysis is a more complex than the ones with only discrete variables, but feasible for a wide class of models. The analytical description of Markov fluid models and a set of solution techniques have been introduced. The potential use of fluid models in performance analysis has been demonstrated by an applicative example.

References

1. M. Agapie and K. Sohraby. Algorithmic solution to second order fluid flow. In *Proc. of IEEE Infocom*, Anchorage, Alaska, Usa, Apr 2001.
2. Soohan Ahn and V. Ramaswami. Matrix-geometric algorithms for stochastic fluid flows. In *SMCtools '06: Proceeding from the 2006 workshop on Tools for solving sturctured Markov chains*, page 11, New York, NY, USA, 2006. ACM Press.
3. M. Ajmone Marsan, G. Balbo, A. Bobbio, G. Chiola, G. Conte, and A. Cumani. The effect of execution policies on the semantics and analysis of stochastic Petri nets. *IEEE Transactions on Software Engineering*, 15(7):832–846, July 1989.
4. M. Ajmone Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis. *Modelling with Generalized Stochastic Petri Nets.* John Wiley & Sons, 1995.
5. H. Alla and R. David. Continuous and Hybrid Petri Nets. *Journal of Systems Circuits and Computers*, 8(1), Feb 1998.
6. Eu-Jin Ang and Javier Barria. The markov modulated regulated brownian motion: A second-order fluid flow model of a finite buffer. *Queueing Systems*, 35:263–287, 2000.
7. D. Anick, D. Mitra, and M. M. Sondhi. Stochastic Theory of a Data-Handling System. *Bell Sys. Thech. J*, 61(8):1871–1894, Oct 1982.
8. D.-Y. Chen, Y. Hong, and K. S. Trivedi. Second order stochastic fluid flow models with fluid dependent flow rates. *Performance Evaluation*, 49(1-4):341–358, 2002.
9. G. Ciardo, D. M. Nicol, and K. S. Trivedi. Discrete-event Simulation of Fluid Stochastic Petri Nets. In *Proc. 7th Int. Workshop on Petri Nets and Performance Models (PNPM'97)*, pages 217–225, Saint Malo, France, June 1997. IEEE Comp. Soc. Press.
10. G. Ciardo, D. M. Nicol, and K. S. Trivedi. Discrete-event Simulation of Fluid Stochastic Petri Nets. *IEEE Transactions on Software Engineering*, 2(25):207–217, 1999.
11. T. Czachorski and F. Pekergin. Diffusion approximation as a modelling tool. In *Tutorial Papers of the ATM & IP 2000*, pages 18/1 – 18/40, Ilkley, UK, Jul 2000.

12. E. de Souza e Silva and R. Gail. An algorithm to calculate transient distributions of cummulative rate and impulse based reward. *Commun. in Statist. – Stochastic Models*, 14(3):509–536, 1998.
13. A. I. Elwalid and D. Mitra. Statistical Multiplexing with Loss Priorities in Rate-Based Congestion Control of High-Speed Networks. *IEEE Transaction on Communications*, 42(11):2989–3002, November 1994.
14. R. Gaeta, M. Gribaudo, D. Manini, and M. Sereno. Analysis of resource transfers in peer-to-peer file sharing applications using fluid models. *Perform. Eval.*, 63(3):149–174, 2006.
15. R. German. *Performance Analysis of Communication Systems: Modeling with Non-Markovian Stochastic Petri Nets*. John Wiley & Sons, 2000.
16. R. German, M. Gribaudo, G. Horváth, and M. Telek. Stationary analysis of FSPNs with mutually dependent discrete and continuous parts. In *International Conference on Petri Net Performance Models – PNPM 2003*, pages 30–39, Urbana, IL, USA, Sept 2003. IEEE CS Press.
17. R. German and C. Lindemann. Analysis of Stochasitc Petri Nets by the Method of Supplementary Variables. *Performance Evaluation*, 20:317–335, 1994.
18. M. Gribaudo and R. German. Numerical solution of bounded fluid models using matrix exponentiation. In *Proc. 11th GI/ITG Conference on Measuring, Modelling and Evaluation of Computer and Communication Systems (MMB)*, Aachen, Germany, Sep 2001. VDE Verlag.
19. M. Gribaudo, M. Sereno, and A. Bobbio. Fluid Stochastic Petri Nets: An Extended Formalism to Include non-Markovian Models. In *Proc.* 8^{th} *Intern. Workshop on Petri Nets and Performance Models*, Zaragoza, Spain, Sep 1999. IEEE-CS Press.
20. M. Gribaudo, M. Sereno, A. Bobbio, and A. Horvath. Fluid Stochastic Petri Nets augmented with Flush-out arcs: Modelling and Analysis. *Discrete Event Dynamic Systems*, 11(1 & 2), 2001.
21. G. Horton, V. G. Kulkarni, D. M. Nicol, and K. S. Trivedi. Fluid stochastic Petri Nets: Theory, Application, and Solution Techniques. *European Journal of Operations Research*, 105(1):184–201, Feb 1998.
22. R. L. Karandikar and V.G. Kulkarni. Second-order fluid flow models: reflected brownian motion in a random environment. *Operations Research*, 43:77–88, 1995.
23. V. G. Kulkarni. Fluid models for single buffer systems. In J. H. Dshalalow, editor, *Models and Applications in Science and Engineering*, Frontiers in Queueing, pages 321–338. CRC Press, 1997.
24. V.G. Kulkarni, V.F. Nicola, and K. Trivedi. Effects of checkpointing and queueing on program performance. *Stochastic models*, 4(6):615–648, 1990.
25. P. A. W. Lewis and G. S. Shedler. Simulation of nonhomogeneous Poisson processes by thinning. *Naval Research Logistic Quarterly*, 26:403–414, 1979.
26. Allam M. *Sur l'analyse quantitative des reseaux de Petri hybrides Une approche basee sur les automates hybrides*. Laboratoire d'Automatique de Grenoble, 1998.
27. D. Manini and M. Gribaudo. Modelling search, availability, and parallel download in p2p file sharing applications with fluid model. *Preecedings of the 14th International Conference on Advanced Computing and Communication ADCOM*, 2006.
28. D. Mitra. Stochastic Theory of a Fluid Model of Producers and Consumers. *Adv. Appl. Prob.*, 20:646–676, 1988.
29. V.F. Nicola, R. Martini, and P.F. Chimento. The completion time of a job in a failure environment and partial loss of work. In *2nd Int. Conf. on Mathematical Methods in Reliability (MMR'2000)*, pages 813–816, Bordeaux, France, July 2000.

30. J.A. Pouwelse, P. Garbacki, D.H.G. Epema, and H.J. Sips. A measurement study of the bittorrent peer-to-peer file-sharing system. *Proc. 19th IEEE Annual Computer Communications Workshop*, 2004.
31. M. A. Qureshi and W. H. Sanders. Reward model solution methods with impulse and rate rewards: An algorithm and numerical results. *Performance Evaluation*, pages 413–436, 1994.
32. A. Reibman, R. Smith, and K.S. Trivedi. Markov and Markov reward model transient analysis: an overview of numerical approaches. *European Journal of Operational Research*, 40:257–267, 1989.
33. Qiang Ren and Hisashi Kobayashi. Transient solutions for the buffer behavior in statistical multiplexing. *Perform. Eval.*, 23(1):65–87, 1995.
34. B. Sericola. Transient analysis of stochastic fluid models. *Perform. Eval.*, 32(4):245–263, 1998.
35. Bruno Sericola. A finite buffer fluid queue driven by a markovian queue. *Queueing Syst. Theory Appl.*, 38(2):213–220, 2001.
36. Bruno Sericola and Bruno Tuffin. A fluid queue driven by a markovian queue. *Queueing Syst. Theory Appl.*, 31(3-4):253–264, 1999.
37. K. Trivedi and V. Kulkarni. FSPNs: Fluid Stochastic Petri nets. In *Application and Theory of Petri Nets 1993, Proc.* 14^{th} *Intern. Conference*, LNCS, Chicago, USA, June 1993. Springer Verlag.
38. K. Wolter. Second order fluid stochastic petri nets: an extension of gspns for approximate and continuous modelling. In *Proc. of World Congress on System Simulation*, pages 328–332, Singapore, Sep 1997.
39. K. Wolter. Jump Transitions in Second Order FSPNs. In *Proc. of MASCOTS'99*, Washington, DC, Oct 1999.
40. K. Wolter, G. Horton, and R. German. Non-Markovian Fluid Stochastic Petri Nets. Technical report, Tecnical University of Berlin, Berlin, Germany, 1996. Report 1996-13.

Tackling Large State Spaces in Performance Modelling*

William J. Knottenbelt and Jeremy T. Bradley

Department of Computing, Imperial College London, Huxley Building, South Kensington, London SW7 2AZ, UK
{wjk,jb}@doc.ic.ac.uk

Abstract. Stochastic performance models provide a powerful way of capturing and analysing the behaviour of complex concurrent systems. Traditionally, performance measures for these models are derived by generating and then analysing a (semi-)Markov chain corresponding to the model's behaviour at the state-transition level. However, and especially when analysing industrial-scale systems, workstation memory and compute power is often overwhelmed by the sheer number of states.

This chapter explores an array of techniques for analysing stochastic performance models with large state spaces. We concentrate on explicit techniques suitable for unstructured state spaces and show how memory and run time requirements can be reduced using a combination of probabilistic algorithms, disk-based solution techniques and communication-efficient parallelism based on hypergraph-partitioning. We apply these methods to different kinds of performance analysis, including steady-state and passage-time analysis, and demonstrate them on case study examples.

1 Introduction and Context

Modern computer and communication systems are increasingly complex. Whereas in the past systems were usually controlled by a single program running on a single machine with a single flow of control, recent years have seen the rise of technologies such as multi-threading, parallel and distributed computing and advanced communication networks. The result is that modern systems are complex webs of cooperating subsystems with many possible interactions.

In the face of this complexity, it is an extremely challenging task for system designers to guarantee satisfactory system operation in terms of both correctness and performance. Unfortunately, attempts to predict dynamic behaviour using intuition or "rules of thumb" are doomed to failure because designers cannot foresee the many millions of possible interactions between components. Likewise, *ad hoc* testing cannot expose a sufficient number of execution paths. Consequently the likelihood of problems caused by subtle bugs such as race conditions is high.

One way to meet the above challenge using a rigorous engineering approach is to use formal modelling techniques to mechanically verify correctness and

* Based on work carried out in collaboration with Nicholas J. Dingle, Peter G. Harrison and Aleksandar Trifunovic.

M. Bernardo and J. Hillston (Eds.): SFM 2007, LNCS 4486, pp. 318–370, 2007.

performance properties. The advantage of this style of approach over *ad hoc* methods has been clearly demonstrated in recent work on the formal model checking of file system code – in [1] the authors use a breadth-first state space exploration of all possible execution paths and failure points to automatically uncover several (serious and hitherto undiscovered) errors in ten widely-used file systems.

Formal techniques which consider all possible system behaviours can likewise be brought to bear on the problem which is the primary concern of the present chapter, namely that of predicting system performance. Our specific focus is on analytical performance modelling techniques which make use of Markov and semi-Markov chains to model the low-level stochastic behaviour of a system. (Semi-)Markov chains are limited to describing systems that have discrete states and which satisfy the property that the future behaviour of the system depends only on the current state. Despite these limitations, they are flexible enough to model many phenomena found in complex concurrent systems such as blocking, synchronisation, preemption, state-dependent routing and complex traffic arrival processes. In addition, tedious manual enumeration of all possible system states is not necessary. Instead, chains can be automatically derived from several widely-used high level modelling formalisms such as Stochastic Petri Nets and Stochastic Process Algebras.

A major difficulty often encountered with this approach is the *state space explosion problem* whereby workstation memory and compute power are overwhelmed by the sheer number of states that emerge from complex models. Consequently, a major challenge and focus of research is the development of methods and data structures which minimise the memory and runtime required to generate and solve very large (semi-)Markov chains. One approach to this "largeness" problem is to restrict the structure of models that can be analysed. This allows for the application of efficient techniques which exploit the restricted structure. Since these techniques are covered in other chapters, we do not discuss them further here, preferring unrestricted scalable parallel and distributed algorithms which are able to efficiently leverage the compute power, memory and disk space of several processors.

2 Stochastic Processes

At the lowest level, the performance modelling of a system can be accomplished by identifying all possible configurations, or *states*, that the system can enter and describing the ways in which the system can move between those states. This is termed the *state-transition* level behaviour of the model, and the changes in state as time progresses describe a *stochastic process*. We focus on those stochastic processes which belong to the class known as *Markov processes*, specifically continuous-time Markov chains (CTMCs) and the more general semi-Markov processes (SMPs).

Consider a random variable χ which takes on different values at different times t. The sequence of random variables $\chi(t)$ is said to be a stochastic process. The

different values which $\chi(t)$ can take, describe the state space of the stochastic process.

A stochastic process can be classified by the nature of its state space and of its time parameter. If the values in the state space of $\chi(t)$ are finite or countably infinite, then the stochastic process is said to have a *discrete state space* (and may also be referred to as a *chain*). Otherwise, the state space is said to be *continuous*. Similarly, if the times at which $\chi(t)$ is observed are also countable, the process is said to be a *discrete-time* process. Otherwise, the process is said to be a *continuous-time* process. In this chapter, all stochastic processes considered have discrete and finite state spaces, and we focus mainly on those which evolve in continuous time.

Definition 1. *A* Markov *process is a stochastic process in which the* Markov property *holds. Given that $\chi(t) = x_t$ indicates that the state of the process $\chi(t)$ at time t is x_t, this property stipulates that:*

$$\begin{aligned}\mathbb{P}\big(\chi(t) = x \mid \chi(t_n) = x_n, \chi(t_{n-1}) = x_{n-1}, \ldots, \chi(t_0) = x_0\big) \\ = \mathbb{P}\big(\chi(t) = x \mid \chi(t_n) = x_n\big) \\ \text{for } t > t_n > t_{n-1} > \ldots > t_0\end{aligned}$$

That is, the future evolution of the system depends only on the current state and not on any prior states.

Definition 2. *A Markov process is said to be* homogeneous *if it is invariant to shifts in time:*

$$\mathbb{P}\big(\chi(t+s) = x \mid \chi(t_n + s) = x_n\big) = \mathbb{P}\big(\chi(t) = x \mid \chi(t_n) = x_n\big)$$

2.1 Continuous-Time Markov Chains

There exists a family of Markov processes with discrete state spaces but whose transitions can occur at arbitrary points in time; we call these continuous-time Markov chains (CTMCs). An homogeneous N-state CTMC has state at time t denoted $\chi(t)$. Its evolution is described by an $N \times N$ generator matrix $\mathbf{Q}$, where q_{ij} is the infinitesimal rate of moving from state i to state j $(i \neq j)$, and $q_{ii} = -\sum_{j \neq i} q_{ij}$.

The Markov property imposes a *memoryless* restriction on the distribution of the sojourn times of states in a CTMC. The future evolution of the system therefore does not depend on the evolution of the system up until the current state, nor does it depend on how long the system has already been in the current state. This means that the sojourn time ν in any state must satisfy:

$$\mathbb{P}(\nu \geq s + t \mid \nu \geq t) = \mathbb{P}(\nu \geq s) \tag{1}$$

A consequence of Eq. (1) is that all sojourn times in a CTMC must be exponentially distributed (see [2] for a proof that this is the only continuous distribution function which satisfies this condition). The rate out of state i, and therefore

the parameter of the sojourn time distribution, is μ_i and is equal to the sum of all rates out of state i, that is $\mu_i = -q_{ii}$. This means that the density function of the sojourn time in state i is $f_i(t) = \mu_i\, e^{-\mu_i t}$ and the average sojourn time in state i is μ_i^{-1}.

A concept that is fundamental to reasoning about the performance of a CTMC is that of its steady state distribution – that is the long-run average proportion of time that a system spends in each of its states.

Definition 3. *A Markov chain is said to be* irreducible *if every state communicates with every other state, i.e. if for every pair of states i and j there is a path from state i to j and vice versa.*

Definition 4. *The* steady-state probability distribution $\{\pi_j\}$ *of an irreducible, homogeneous CTMC is given by:*

$$\pi_j = \lim_{t\to\infty} \mathbb{P}(\chi(t) = j \mid \chi(0) = i)$$

For a finite, irreducible and homogeneous CTMC, the steady-state probabilities $\{\pi_j\}$ always exist and are independent of the initial state distribution. They are uniquely given by the solution of the equations:

$$-q_{jj}\pi_j + \sum_{k\neq j} q_{kj}\pi_k = 0 \;\text{ subject to }\; \sum_i \pi_i = 1$$

Again, this can be expressed in matrix vector form (in terms of the vector $\boldsymbol{\pi}$ with elements $\{\pi_1, \pi_2, \ldots, \pi_N\}$ and the matrix $\mathbf{Q}$ defined above) as:

$$\boldsymbol{\pi}\mathbf{Q} = \mathbf{0} \tag{2}$$

A CTMC also has an embedded discrete-time Markov chain (EMC) which describes the behaviour of the chain at state-transition instants, that is to say the probability that the next state is j given that the current state is i. The EMC of a CTMC has a one-step $N \times N$ transition matrix $\mathbf{P}$ where $p_{ij} = -q_{ij}/q_{ii}$ for $i \neq j$ and $p_{ij} = 0$ for $i = j$.

The steady-state distribution enables us to compute various basic resource-based measures (such as utilisation, mean throughput, and so on); however, more advanced response-time measures (such as quantiles of response time) require a *first passage time* analysis.

Definition 5. *Consider a finite, irreducible CTMC with N states $\{1, 2, \ldots, N\}$ and generator matrix $\mathbf{Q}$. If $\chi(t)$ denotes the states of the CTMC at time t ($t \geq 0$) and $N(t)$ denotes the number of state transitions which have occurred by time t, the* first passage time *from a single source marking i into a non-empty set of target markings $\boldsymbol{j}$ is:*

$$P_{i\boldsymbol{j}}(t) = \inf\{u > 0 : \chi(t+u) \in \boldsymbol{j}, N(t+u) > N(t), \chi(t) = i\}$$

When the CTMC is stationary and time-homogeneous this quantity is independent of t:

$$P_{i\boldsymbol{j}} = \inf\{u > 0 : \chi(u) \in \boldsymbol{j}, N(u) > 0, \chi(0) = i\} \tag{3}$$

That is, the first time the system enters a state in the set of target states $\boldsymbol{j}$, given that the system began in the source state i and at least one state transition has occurred. $P_{i\boldsymbol{j}}$ is a random variable with probability density function $f_{i\boldsymbol{j}}(t)$ such that:

$$\mathbb{P}(t_1 < P_{i\boldsymbol{j}} < t_2) = \int_{t_1}^{t_2} f_{i\boldsymbol{j}}(t)\,\mathrm{d}t \qquad \text{for } 0 \leq t_1 < t_2$$

In order to determine $f_{i\boldsymbol{j}}(t)$ it is necessary to convolve the state holding-time density functions over all possible paths (including cycles) from state i to all of the states in $\boldsymbol{j}$.

The calculation of the convolution of two functions in t-space can be more easily accomplished by multiplying their Laplace transforms together in s-space and inverting the result. The calculation of $f_{i\boldsymbol{j}}(t)$ is therefore achieved by calculating the Laplace transform of the convolution of the state holding times over all paths between i and $\boldsymbol{j}$ and then numerically inverting this Laplace transform (see Sect. 4.3 for a description of two inversion algorithms).

In a CTMC all state sojourn times are exponentially distributed, so the density function of the sojourn time in state i is $\mu_i e^{-\mu_i t}$, where $\mu_i = -q_{ii}$ (as before). The Laplace transform of an exponential density function with rate parameter λ is:

$$L\{\lambda e^{-\lambda t}\} = \frac{\lambda}{\lambda + s}$$

Denoting the Laplace transform of the density function $f_{i\boldsymbol{j}}(t)$ of the passage time random variable $P_{i\boldsymbol{j}}$ as $L_{i\boldsymbol{j}}(s)$, we proceed by means of a first-step analysis. That is, to calculate the first passage time from state i into the set of target states $\boldsymbol{j}$, we consider moving from state i to its set of direct successor states $\boldsymbol{k}$ and thence from states in $\boldsymbol{k}$ to states in $\boldsymbol{j}$. This can be expressed as the following system of linear equations:

$$L_{i\boldsymbol{j}}(s) = \sum_{k \notin \boldsymbol{j}} p_{ik} \left(\frac{-q_{ii}}{s - q_{ii}} \right) L_{k\boldsymbol{j}}(s) + \sum_{k \in \boldsymbol{j}} p_{ik} \left(\frac{-q_{ii}}{s - q_{ii}} \right) \tag{4}$$

The first term (i.e. the summation over non-target states $k \notin \boldsymbol{j}$) convolves the sojourn time density in state i with the density of the time taken for the system to evolve from state k into a target state in $\boldsymbol{j}$, weighted by the probability that the system transits from state i to state k. The second term (i.e. the summation over target states $k \in \boldsymbol{j}$) simply reflects the sojourn time density in state i weighted by the probability that a transition from state i into a target state k occurs.

Given that $p_{ij} = -q_{ij}/q_{ii}$ in the context of a CTMC, Eq. (4) can be rewritten more simply as:

$$L_{i\boldsymbol{j}}(s) = \sum_{k \notin \boldsymbol{j}} \frac{q_{ik}}{s - q_{ii}} L_{k\boldsymbol{j}}(s) + \sum_{k \in \boldsymbol{j}} \frac{q_{ik}}{s - q_{ii}} \tag{5}$$

This set of linear equations can be expressed in matrix–vector form. For example, when $\boldsymbol{j} = \{1\}$ we have:

$$\begin{pmatrix} s - q_{11} & -q_{12} & \cdots & -q_{1n} \\ 0 & s - q_{22} & \cdots & -q_{2n} \\ 0 & -q_{32} & \cdots & -q_{3n} \\ 0 & \vdots & \ddots & \vdots \\ 0 & -q_{n2} & \cdots & s - q_{nn} \end{pmatrix} \begin{pmatrix} L_{1\boldsymbol{j}}(s) \\ L_{2\boldsymbol{j}}(s) \\ L_{3\boldsymbol{j}}(s) \\ \vdots \\ L_{n\boldsymbol{j}}(s) \end{pmatrix} = \begin{pmatrix} 0 \\ q_{21} \\ q_{31} \\ \vdots \\ q_{n1} \end{pmatrix} \tag{6}$$

Our formulation of the passage time quantity in Eq. (3) states that we must observe at least one state-transition during the passage. In the case where $i \in \boldsymbol{j}$ (as for $L_{1\boldsymbol{j}}(s)$ in the above example), we therefore calculate the density of the cycle time to return to state i rather than requiring $L_{i\boldsymbol{j}}(s) = 1$.

Given a particular (complex-valued) s, Eq. (5) can be solved for $L_{i\boldsymbol{j}}(s)$ by standard iterative numerical techniques for the solution of systems of linear equations in $\mathbf{Ax} = \mathbf{b}$ form. Many numerical Laplace transform inversion algorithms (such as the Euler and Laguerre methods) can identify in advance the s-values at which $L_{i\boldsymbol{j}}(s)$ must be calculated in order to perform the numerical inversion. Therefore, if the algorithm requires m different values of $L_{i\boldsymbol{j}}(s)$, Eq. (5) will need to be solved m times.

The corresponding cumulative distribution function $F_{i\boldsymbol{j}}(t)$ of the passage time is obtained by integrating under the density function. This integration can be achieved in terms of the Laplace transform of the density function by dividing it by s, i.e. $F^*_{i\boldsymbol{j}}(s) = L_{i\boldsymbol{j}}(s)/s$. In practice, if Eq. (5) is solved as part of the inversion process for calculating $f_{i\boldsymbol{j}}(t)$, the m values of $L_{i\boldsymbol{j}}(s)$ can be retained. Once the numerical inversion algorithm has used them to compute $f_{i\boldsymbol{j}}(t)$, these values can be recovered, divided by s and then taken as input by the numerical inversion algorithm again to compute $F_{i\boldsymbol{j}}(t)$. Thus, in calculating $f_{i\boldsymbol{j}}(t)$, we get $F_{i\boldsymbol{j}}(t)$ for little further computational effort.

When there are multiple source markings, denoted by the vector $\boldsymbol{i}$, the Laplace transform of the response time density at equilibrium is:

$$L_{\boldsymbol{i}\boldsymbol{j}}(s) = \sum_{k \in \boldsymbol{i}} \alpha_k L_{k\boldsymbol{j}}(s)$$

where the weight α_k is the equilibrium probability that the state is $k \in \boldsymbol{i}$ at the starting instant of the passage. This instant is the moment of entry into state k; thus α_k is proportional to the equilibrium probability of the state k in the underlying embedded (discrete-time) Markov chain (EMC) of the CTMC with one-step transition matrix $\mathbf{P}$ as defined in Sect. 2.1. That is:

$$\alpha_k = \begin{cases} \pi_k / \sum_{j \in \boldsymbol{i}} \pi_j & \text{if } k \in \boldsymbol{i} \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

where the vector $\boldsymbol{\pi}$ is any non-zero solution to $\boldsymbol{\pi} = \boldsymbol{\pi}\mathbf{P}$. The row vector with components α_k is denoted by $\boldsymbol{\alpha}$.

Uniformisation. Passage time densities and quantiles in CTMCs may also be computed through the use of *uniformisation* (also known as *randomisation*) [3,4,5,6,7,8]. This transforms a CTMC into one in which all states have the same mean holding time $1/q$, by allowing "invisible" transitions from a state to itself. This is equivalent to a discrete-time Markov chain, after normalisation of the rows, together with an associated Poisson process of rate q.

Definition 6. *The one-step transition probability matrix* $\mathbf{P}$ *which characterises the one-step behaviour of a uniformised DTMC is derived from the generator matrix* $\mathbf{Q}$ *of the CTMC as:*

$$\mathbf{P} = \mathbf{Q}/q + \mathbf{I} \tag{8}$$

where the rate $q > \max_i |q_{ii}|$ *ensures that the DTMC is aperiodic by guaranteeing that there is at least one single-step transition from a state to itself.*

We ensure that only the first passage time density is calculated and that we do not consider the case of successive visits to a target state by making the target states in $\mathbf{P}$ absorbing. We denote by $\mathbf{P}'$ the one-step transition probability matrix of the modified, uniformised chain.

The calculation of the first passage time density between two states then has two main components. The first considers the time to complete n hops ($n = 1, 2, 3, \ldots$). Recall that in the uniformised chain all transitions occur with rate q. The density of the time taken to move between two states is found by convolving the state holding-time densities along all possible paths between the states. In a standard CTMC, convolving holding times in this manner is non-trivial as, although they are all exponentially distributed, their rate parameters are different. In a CTMC which has undergone uniformisation, however, all states have exponentially-distributed state holding-times with the same parameter q. This means that the convolution of n of these holding-time densities is an n-stage Erlang density with rate parameter q.

Secondly, it is necessary to calculate the probability that the transition between a source and target state occurs in exactly n hops of the uniformised chain, for every value of n between 1 and a maximum value m. The value of m is determined when the value of the nth Erlang density function (the left-hand term in Eq. (9)) drops below some threshold value. After this point, further terms are deemed to add nothing significant to the passage time density and so are disregarded.

The density of the time to pass between a source state i and a target state j in a uniformised Markov chain can therefore be expressed as the sum of m n-stage Erlang densities, weighted with the probability that the chain moves from state i to state j in exactly n hops ($1 \leq n \leq m$). This can be generalised to allow for multiple target states in a straightforward manner; when there are multiple source states it is necessary to provide a probability distribution across this set of states (such as the renormalised steady-state distribution calculated below in Eq. (11)).

The response time between the non-empty set of source states $\boldsymbol{i}$ and the non-empty set of target states $\boldsymbol{j}$ in the uniformised chain therefore has probability density function:

$$f_{\boldsymbol{ij}}(t) = \sum_{n=1}^{\infty} \left(\frac{q^n t^{n-1} e^{-qt}}{(n-1)!} \sum_{k \in \boldsymbol{j}} \pi_k^{(n)} \right)$$
$$\simeq \sum_{n=1}^{m} \left(\frac{q^n t^{n-1} e^{-qt}}{(n-1)!} \sum_{k \in \boldsymbol{j}} \pi_k^{(n)} \right) \tag{9}$$

where:

$$\boldsymbol{\pi}^{(n+1)} = \boldsymbol{\pi}^{(n)} \mathbf{P}' \qquad \text{for } n \geq 0 \tag{10}$$

with:

$$\pi_k^{(0)} = \begin{cases} 0 & \text{for } k \notin \boldsymbol{i} \\ \pi_k / \sum_{j \in \boldsymbol{i}} \pi_j & \text{for } k \in \boldsymbol{i} \end{cases} \tag{11}$$

The π_k values are the steady state probabilities of the corresponding state k in the CTMC's embedded Markov chain. When the convergence criterion:

$$\frac{\|\boldsymbol{\pi}^{(n)} - \boldsymbol{\pi}^{(n-1)}\|_\infty}{\|\boldsymbol{\pi}^{(n)}\|_\infty} < \varepsilon \tag{12}$$

is met, for given tolerance ε, the vector $\boldsymbol{\pi}^{(n)}$ is considered to have converged and no further multiplications with $\mathbf{P}'$ are performed. Here, $\|\mathbf{x}\|_\infty$ is the infinity-norm given by $\|\mathbf{x}\|_\infty = \max_i |x_i|$.

The corresponding cumulative distribution function for the passage time, $F_{\boldsymbol{ij}}(t)$, can be calculated by substituting the cumulative distribution function for the Erlang distribution into Eq. (9) in place of the Erlang density function term, viz.:

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

where $\boldsymbol{\pi}^{(n)}$ is defined as in Eqs. (10) and (11).

2.2 Semi-Markov Processes

Semi-Markov Processes (SMPs) are an extension of Markov processes which allow for generally distributed sojourn times. Although the memoryless property no longer holds for state sojourn times, at transition instants SMPs still behave in the same way as Markov processes (that is to say, the choice of the next state is based only on the current state) and so share some of their analytical tractability.

Definition 7. *Consider a Markov renewal process* $\{(\chi_n, T_n) : n \geq 0\}$ *where* T_n *is the time of the* n*th transition (*$T_0 = 0$*) and* $\chi_n \in \mathcal{S}$ *is the state at the* n*th transition. Let the kernel of this process be:*

$$R(n, i, j, t) = \mathbb{P}(\chi_{n+1} = j, T_{n+1} - T_n \leq t \mid \chi_n = i)$$

for $i, j \in \mathcal{S}$*. The continuous time semi-Markov process,* $\{Z(t), t \geq 0\}$*, defined by the kernel* R*, is related to the Markov renewal process by:*

$$Z(t) = \chi_{N(t)}$$

where $N(t) = \max\{n : T_n \leq t\}$*, i.e. the number of state transitions that have taken place by time* t*. Thus* $Z(t)$ *represents the state of the system at time* t*.*

We consider only time-homogeneous SMPs in which $R(n, i, j, t)$ is independent of n, that is for:

$$\begin{aligned} R(i, j, t) &= \mathbb{P}(\chi_{n+1} = j, T_{n+1} - T_n \leq t \mid \chi_n = i) \quad \text{for any } n \geq 0 \\ &= p_{ij} H_{ij}(t) \end{aligned}$$

where $p_{ij} = \mathbb{P}(\chi_{n+1} = j \mid \chi_n = i)$ is the state transition probability between states i and j and $H_{ij}(t) = \mathbb{P}(T_{n+1} - T_n \leq t \mid \chi_{n+1} = j, \chi_n = i)$, is the sojourn time distribution in state i when the next state is j. An SMP can therefore be characterised by two matrices $\mathbf{P}$ and $\mathbf{H}$ with elements p_{ij} and H_{ij} respectively.

Semi-Markov processes can be analysed for steady-state performance metrics in a similar manner as DTMCs and CTMCs. To do this, we need to know the steady-state probabilities of the SMP's embedded Markov chain and the average time spent in each state. The first of these can be calculated by solving $\boldsymbol{\pi} = \boldsymbol{\pi}\mathbf{P}$, as in the case of DTMCs. The average time in state i, $\mathbb{E}[\tau_i]$, is the weighted sum of the averages of the sojourn time in the state i when going to state j, $\mathbb{E}[\tau_{ij}]$, for all successor states j of i, that is:

$$\mathbb{E}[\tau_i] = \sum_j p_{ij} \mathbb{E}[\tau_{ij}]$$

The steady-state probability of being in state i of the SMP is then:

$$\phi_i = \frac{\pi_i \mathbb{E}[\tau_i]}{\sum_{m=1}^{N} \pi_m \mathbb{E}[\tau_m]} \tag{13}$$

That is, the long-run probability of finding the SMP in state i is the probability of its EMC being in state i multiplied by the average amount of time the SMP spends in state i, normalised over the mean total time spent in all of the states of the SMP.

Passage-time analysis for SMPs is also possible by extending the Laplace transform method for CTMCs to cater for generally-distributed state sojourn times.

Definition 8. *Consider a finite, irreducible, continuous-time semi-Markov process with N states $\{1, 2, \ldots, N\}$. Recalling that $Z(t)$ denotes the state of the SMP at time t ($t \geq 0$), the first passage time from a source state i at time t into a non-empty set of target states $\boldsymbol{j}$ is:*

$$P_{i\boldsymbol{j}}(t) = \inf\{u > 0 : Z(t+u) \in \boldsymbol{j}, N(t+u) > N(t) \mid Z(t) = i\} \tag{14}$$

For a stationary time-homogeneous SMP, $P_{i\boldsymbol{j}}(t)$ is independent of t and we have:

$$P_{i\boldsymbol{j}} = \inf\{u > 0 : Z(u) \in \boldsymbol{j}, N(u) > 0 \mid Z(0) = i\} \tag{15}$$

$P_{i\boldsymbol{j}}$ has an associated probability density function $f_{i\boldsymbol{j}}(t)$ such that the passage time quantile is given as:

$$\mathbb{P}(t_1 < P_{i\boldsymbol{j}} < t_2) = \int_{t_1}^{t_2} f_{i\boldsymbol{j}}(t)\,\mathrm{d}t \qquad \text{for } 0 \leq t_1 < t_2 \tag{16}$$

In general, the Laplace transform of $f_{i\boldsymbol{j}}$, $L_{i\boldsymbol{j}}(s)$, can be computed by solving a set of N linear equations:

$$L_{i\boldsymbol{j}}(s) = \sum_{k \notin \boldsymbol{j}} r^*_{ik}(s) L_{k\boldsymbol{j}}(s) + \sum_{k \in \boldsymbol{j}} r^*_{ik}(s) \qquad \text{for } 1 \leq i \leq N \tag{17}$$

where $r^*_{ik}(s)$ is the Laplace-Stieltjes transform (LST) of $R(i,k,t)$ and is defined by:

$$r^*_{ik}(s) = \int_0^\infty e^{-st}\,\mathrm{d}R(i,k,t) \tag{18}$$

Eq. (17) has a matrix–vector form where the elements of the matrix are arbitrary complex functions; care needs to be taken when storing such functions for eventual numerical inversion (see Sect. 4.3). For example, when $\boldsymbol{j} = \{1\}$, Eq. (17) yields:

$$\begin{pmatrix} 1 & -r^*_{12}(s) & \cdots & -r^*_{1N}(s) \\ 0 & 1 - r^*_{22}(s) & \cdots & -r^*_{2N}(s) \\ 0 & -r^*_{32}(s) & \cdots & -r^*_{3N}(s) \\ \vdots & \vdots & \ddots & \vdots \\ 0 & -r^*_{N2}(s) & \cdots & 1 - r^*_{NN}(s) \end{pmatrix} \begin{pmatrix} L_{1\boldsymbol{j}}(s) \\ L_{2\boldsymbol{j}}(s) \\ L_{3\boldsymbol{j}}(s) \\ \vdots \\ L_{N\boldsymbol{j}}(s) \end{pmatrix} = \begin{pmatrix} r^*_{11}(s) \\ r^*_{21}(s) \\ r^*_{31}(s) \\ \vdots \\ r^*_{N1}(s) \end{pmatrix} \tag{19}$$

When there are multiple source states, denoted by the vector $\boldsymbol{i}$, the Laplace transform of the passage time density at steady-state is:

$$L_{\boldsymbol{i}\boldsymbol{j}}(s) = \sum_{k \in \boldsymbol{i}} \alpha_k L_{k\boldsymbol{j}}(s) \tag{20}$$

where the weight α_k is the probability at equilibrium that the system is in state $k \in \boldsymbol{i}$ at the starting instant of the passage. As with CTMCs $\boldsymbol{\alpha}$ is defined in terms of $\boldsymbol{\pi}$, the steady-state vector of the embedded discrete-time Markov chain with one-step transition probability matrix $\mathbf{P}$:

$$\alpha_k = \begin{cases} \pi_k / \sum_{j \in \boldsymbol{i}} \pi_j & \text{if } k \in \boldsymbol{i} \\ 0 & \text{otherwise} \end{cases} \tag{21}$$

3 Modelling Formalisms

Stochastic models are specified using graphical or symbolic languages known as *modelling formalisms*. Below we describe two popular formalisms: Stochastic Petri nets and Stochastic Process Algebras.

3.1 Stochastic Petri Nets

We briefly outline two types of stochastic Petri net: Generalised Stochastic Petri Nets (GSPNs) which allow timed exponential and immediate transitions, and Semi-Markov Stochastic Petri Nets (SM-SPNs) which specify models with generally distributed transitions.

Generalised Stochastic Petri Nets. Generalised Stochastic Petri nets are an extension of Place-Transition nets, which are ordinary, untimed Petri nets. A Place-Transition net does not have firing delays associated with its transitions and is formally defined in [2]:

Definition 9. *A* Place-Transition net *is a 5-tuple* $PN = (P, T, I^-, I^+, M_0)$ *where*

- $P = \{p_1, \ldots, p_n\}$ *is a finite and non-empty set of places.*
- $T = \{t_1, \ldots, t_m\}$ *is a finite and non-empty set of transitions.*
- $P \cap T = \emptyset$.
- $I^-, I^+ : P \times T \to \mathbb{N}_0$ *are the backward and forward incidence functions, respectively. If* $I^-(p,t) > 0$*, an arc leads from place* p *to transition* t*, and if* $I^+(p,t) > 0$ *then an arc leads from transition* t *to place* p*.*
- $M_0 : P \to \mathbb{N}_0$ *is the initial marking defining the initial number of tokens on every place.*

A marking is a vector of integers representing the number of tokens on each place in a Petri net. The set of all markings that are reachable from the initial marking M_0 is known as the *state space* or *reachability set* of the Petri net, and is denoted by $R(M_0)$. The connections between markings in the reachability set form the *reachability graph*. Formally, if the firing of a transition that is enabled in marking M_i results in marking M_j, then the reachability graph contains a directed arc from marking M_i to marking M_j.

GSPNs [9] are timed extensions of Place-Transition nets with two types of transitions: *immediate* transitions and *timed* transitions. Once enabled, immediate transitions fire in zero time, while timed transitions fire after an exponentially distributed firing delay. Firing of immediate transitions has priority over the firing of timed transitions.

The formal definition of a GSPN is as follows [2]:

Definition 10. *A GSPN is a 4-tuple* $GSPN = (PN, T_1, T_2, W)$ *where*

- $PN = (P, T, I^-, I^+, M_0)$ *is the underlying Place-Transition net.*
- $T_1 \subseteq T$ *is the set of timed transitions,* $T_1 \neq \emptyset$*,*

- $T_2 \subset T$ *denotes the set of immediate transitions,* $T_1 \cap T_2 = \emptyset$, $T = T_1 \cup T_2$
- $W = (w_1, \ldots, w_{|T|})$ *is an array whose entry* w_i *is either*
  - *a (possibly marking dependent)* **rate** $\in \mathbb{R}^+$ *of an exponential distribution specifying the firing delay, when transition* t_i *is a timed transition, i.e.* $t_i \in T_1$
    or
  - *a (possibly marking dependent)* **weight** $\in \mathbb{R}^+$ *specifying the relative firing frequency, when transition* t_i *is an immediate transition, i.e.* $t_i \in T_2$.

The reachability graph of a GSPN contains two types of markings. A vanishing marking is one in which an immediate transition is enabled. The sojourn time in such markings is zero. A tangible marking is one which enables only timed transitions. The sojourn time in such markings is exponentially distributed. Once vanishing markings have been eliminated (see [10] for a discussion of methods for vanishing state elimination), the resulting tangible reachability graph of a GSPN maps directly onto a CTMC.

Semi-Markov Stochastic Petri Nets. Semi-Markov stochastic Petri nets [11] (SM-SPNs) are extensions of GSPNs which support arbitrary holding-time distributions and which generate an underlying semi-Markov process rather than a Markov process. Note that it is not intended that they be a novel technique for dealing with concurrently-enabled generally-distributed transitions. They are instead a useful high-level vehicle for the construction of large semi-Markov models for analysis.

Definition 11. *An SM-SPN consists of a 4-tuple,* $(PN, \mathcal{P}, \mathcal{W}, \mathcal{D})$, *where:*

- $PN = (P, T, I^-, I^+, M_0)$ *is the underlying Place-Transition net.* P *is the set of places,* T, *the set of transitions,* $I^{+/-}$ *are the forward and backward incidence functions describing the connections between places and transitions and* M_0 *is the initial marking.*
- $\mathcal{P} : T \times \mathcal{M} \to \mathbb{Z}^+$, *denoted* $p_t(m)$, *is a marking-dependent priority function for a transition.*
- $\mathcal{W} : T \times \mathcal{M} \to \mathbb{R}^+$, *denoted* $w_t(m)$, *is a marking-dependent weight function for a transition, to allow implementation of probabilistic choice.*
- $\mathcal{D} : T \times \mathcal{M} \to (\mathbb{R}^+ \to [0, 1])$, *denoted* $d_t(m)$, *is a marking-dependent cumulative distribution function for the firing time of a transition.*

In the above, $\mathcal{M}$ *is the set of all markings for a given net. Further, we define the following general net-enabling functions:*

- $\mathcal{E}_N : \mathcal{M} \to P(T)$, *a function that specifies net-enabled transitions from a given marking.*
- $\mathcal{E}_P : \mathcal{M} \to P(T)$, *a function that specifies priority-enabled transitions from a given marking.*

The net-enabling function, $\mathcal{E}_N$, is defined in the usual way for standard Petri nets: if all preceding places have occupying tokens then a transition is net-enabled.

Similarly, we define the more stringent priority-enabling function, $\mathcal{E}_P$. For a given marking, m, $\mathcal{E}_P(m)$ selects only those net-enabled transitions that have the highest priority, that is:

$$\mathcal{E}_P(m) = \{t \in \mathcal{E}_N(m) : p_t(m) = \max\{p_{t'}(m) : t' \in \mathcal{E}_N(m)\}\} \tag{22}$$

Now for a given priority-enabled transition, $t \in \mathcal{E}_P(m)$, the probability that it will be the one that actually fires after a delay sampled from its firing distribution, $d_t(m)$, is:

$$\mathbb{P}(t \in \mathcal{E}_P(m) \text{ fires}) = \frac{w_t(m)}{\sum_{t' \in \mathcal{E}_P(m)} w_{t'}(m)} \tag{23}$$

Note that the choice of which priority-enabled transition is fired in any given marking is made by a probabilistic selection based on transition weights, and is not a race condition based on finding the minimum of samples extracted from firing-time distributions. This mechanism enables the underlying reachability graph of an SM-SPN to be mapped directly onto a semi-Markov chain.

3.2 Stochastic Process Algebras

A process algebra is an abstract language which differs from the formalisms we have considered so far because it is not based on a notion of *flow*. Instead, systems are modelled as a collection of cooperating *agents* or *processes* which execute atomic *actions*. These actions can be carried out independently or can be synchronised with the actions of other agents.

Since models are typically built up from smaller components using a small set of combinators, process algebras are particularly suited to the modelling of large systems with hierarchical structure. This support for *compositionality* is complemented by mechanisms to provide abstraction and compositional reasoning.

Two of the best known process algebras are Hoare's Communicating Sequential Processes (CSP) [12] and Milner's Calculus of Communicating Systems (CCS) [13]. These algebras do not include a notion of time so they can only be used to determine qualitative correctness properties of systems such as the freedom from deadlock and livelock. Stochastic Process Algebras (SPAs) associate a random variable, representing a time duration, with each action. This addition allows quantitative performance analysis to be carried out on SPA models in the same fashion as for SPNs.

Here we will briefly describe the Markovian SPA, PEPA [14]. Other SPAs include TIPP [15,16], MPA [17] and EMPA [18] which are similar to PEPA. A detailed comparison of Markovian stochastic process algebras can be found in [19]. More recently developed non-Markovian SPAs allow for generally-distributed delays as part of the model; examples of these include SPADES [20,21], semi-Markov PEPA [22] and iGSMPA [23,24].

PEPA models are built from components which perform activities of form (α, r) where α is the action type and $r \in \mathbb{R}^+ \cup \{\top\}$ is the exponentially distributed rate of the action. The special symbol $\top$ denotes an passive activity that may only take place in synchrony with another action whose rate is specified.

Interaction between components is expressed using a small set of combinators, which are briefly described below:

Action prefix: Given a process P, $(\alpha, r).P$ represents a process that performs an activity of type α, which has a duration exponentially distributed with mean $1/r$, and then evolves into P.

Constant definition: Given a process Q, $P \stackrel{def}{=} Q$ means that P is a process which behaves in exactly the same way as Q.

Competitive choice: Given processes P and Q, $P + Q$ represents a process that behaves either as P or as Q. The current activities of both P and Q are enabled and a race condition determines into which component the process will evolve.

Cooperation: Given processes P and Q and a set of action types L, $P \underset{L}{\bowtie} Q$ defines the concurrent synchronised execution of P and Q over the cooperation set L. No synchronisation takes place for any activity $\alpha \notin L$, so such activities can take place independently. However, an activity $\alpha \in L$ only occurs when both P and Q are capable of performing the action. The rate at which the action occurs is given by the minimum of the rates at which the two components would have executed the action in isolation.
Cooperation over the empty set $P \underset{\emptyset}{\bowtie} Q$ represents the independent concurrent execution of processes P and Q and is denoted by $P \parallel Q$.

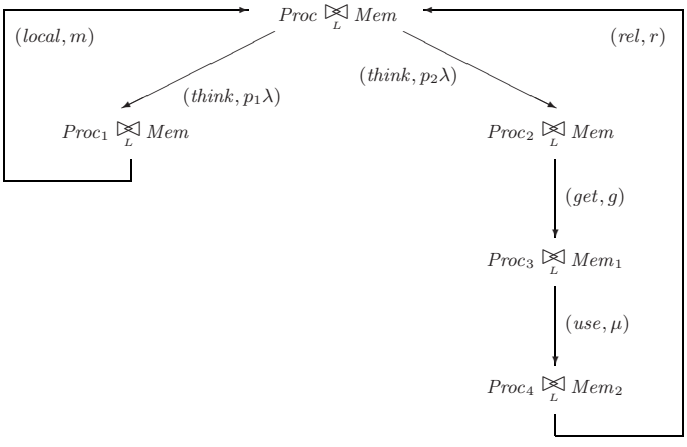
Encapsulation: Given a process P and a set of actions L, P/L represents a process that behaves like P except that activities $\alpha \in L$ are hidden and performed as a *silent* activity. Such activities cannot be part of a cooperation set.

PEPA specifications can be mapped onto continuous time Markov chains in a straightforward manner. Based on the labelled transition system semantics that are normally specified for a process algebra system, a transition diagram or *derivation graph* can be associated with any language expression. This graph describes all possible evolutions of a system and, like a tangible reachability graph in the context of GSPNs, is isomorphic to a CTMC which can be solved for its steady-state distribution. Fig. 1 shows a PEPA specification of a multiprocessor system together with its corresponding derivation graph.

4 Methods for Tackling Large Unstructured State Spaces

We proceed to review several approaches to the problem of analysing stochastic models with large underlying state spaces, covering the major phases in an advanced performance analysis pipeline, i.e. state generation, steady-state solution and passage-time analysis.

The methods reviewed here are based on explicit state representation, and so are particularly suited to the analysis of systems with large unstructured state spaces. We note that there are effective approaches based on implicit/symbolic state representation which can be applied to systems whose underlying state



$$
\begin{array}{llll}
Proc \stackrel{def}{=} (think, p_1\lambda).Proc_1 + (think, p_2\lambda).Proc_2 & & & \\
Proc_1 \stackrel{def}{=} (local, m).Proc & & & \\
Proc_2 \stackrel{def}{=} (get, g).Proc_3 & Proc_3 \stackrel{def}{=} (use, \mu).Proc_4 & Proc_4 \stackrel{def}{=} (rel, r).Proc & \\
Mem \stackrel{def}{=} (get, \top).Mem_1 & Mem_1 \stackrel{def}{=} (use, \mu).Mem_2 & Mem_2 \stackrel{def}{=} (rel, \top).Mem & \\
Sys_4 \stackrel{def}{=} Proc \underset{L}{\bowtie} Mem & \text{where } L = \{get, rel, use\} & &
\end{array}
$$

Fig. 1. A PEPA specification and its corresponding derivation graph [25]

spaces are structured in some way – for example methods based on Binary Decision Diagrams and related data structures [26,27,7,28], and Kronecker methods [29]. Since these methods are the subjects of other chapters in this volume, they are not discussed further here.

4.1 Probabilistic State Space Generation

The first challenge in the quantitative analysis of stochastic models is to generate all reachable states or configurations that the system can enter. The main obstacle to this task is the huge number of states that can emerge, a problem compounded by the large size of individual state descriptors. Consequently there are severe memory and time constraints on the number of states that can be generated using a simplistic explicit exhaustive enumeration.

The Case for Probabilistic Algorithms. A useful, but at first seemingly bizarre, method of dealing with a problem that seems to be infeasible (either in terms of computational or storage demands) is to relax the requirement that a solution should always produce the correct answer. Adopting such a *probabilistic* or *randomised* approach can lead to dramatic memory and time savings. Of course, in order to be useful in practice, the risk of producing an incorrect result must be quantified and kept very small.

One of the most exciting early applications of probabilistic algorithms was in finding an efficient solution to the primality problem (i.e. to determine if some positive integer n is prime). This problem has direct application to public key cryptographic systems, many of which are based on finding a modulus of form pq where p and q are large prime numbers.

The Miller-Rabin primality test [30] provides an efficient probabilistic solution to the primality problem by relying on three facts:

- If n is composite (i.e. not prime) then at least three quarters of the natural numbers less than n are *witnesses* to the compositeness of n (i.e. can be used to establish that n is not prime).
- If n is prime then there is no natural number less than n that is witness to the compositeness of n.
- Given number natural numbers m and n with $m < n$, there is an efficient algorithm which ascertains whether or not m is a witness to the compositeness of n.

The algorithm works by performing k witness tests using randomly chosen natural numbers less than n; should all of these witness tests fail, we assume n is prime. Indeed, if n is prime, this is the correct conclusion. If n is composite, the chances of failing to find a witness (and hence detect that the number is not prime) is 2^{-2k}. Hence, by increasing k, we can arbitrarily increase the reliability of the algorithm at logarithmic run time cost; when k is around 20, the algorithm is probably more reliable than most computer hardware.

Application to State Space Generation. While research into probabilistic algorithms for solving the primality problem has been focused on reducing run time, the application of probabilistic algorithms to state space generation has been focused on the need to reduce memory requirements. In particular, the memory consumption of explicit state space generation algorithms is heavily dependent on the layout and management of a data structure known as the *explored state table*. This table prevents redundant work by identifying which states have already been encountered. Its implementation is particularly challenging because the table is accessed randomly and must be able to rapidly store and retrieve information about every reachable state. One approach is to store the full state descriptor of each state in the table. This *exhaustive* approach guarantees full coverage, but at very high memory cost. *Probabilistic* methods use one-way hashing techniques to drastically reduce the amount of memory required to store states. However, this introduces the risk that two distinct states will have the same hashed representation, resulting in the misidentification and omission of states in the state graph. Naturally, it is important to quantify this risk and to find ways of reducing it to an acceptable level.

The next sections review three of the best-known probabilistic methods (interested readers might also like to consult [31] which presents another recent survey). In each case, we include an analysis and discussion of memory consumption and the omission probability.

Holzmann's Bit-state Hashing. Holzmann's bit-state hashing (or supertrace) technique [32,33] was developed in an attempt to maximize state coverage in the face of limited memory. The technique has proved popular because of its elegance and simplicity and has consequently been included in many research and commercial verification tools.

Holzmann's method is based on the use of Bloom Filters. These were conceived by Burton H. Bloom in 1970 as space-efficient probabilistic data structures for testing set membership [34]. Here the explored state table takes the form of a bit vector T. Initially all bits in T are set to zero. States are mapped into positions in this bit vector using a hash function h, so that when state s is inserted into the table its corresponding bit $T[h(s)]$ is set to one. To check whether a state s is already in the table, the value of $T[h(s)]$ is examined. If it is zero, we know that the state has definitely not been previously encountered; otherwise it is assumed that the state has already been explored. This may be a mistake, however, since two distinct states can be hashed onto the same position in the bit vector. The result of a hash collision will be that one of the states will be incorrectly classified as explored, resulting in the omission of one or more states from the state space. Assuming a good hash function which distributes states randomly, the probability of no hash collisions p when inserting n states into a bit vector of t bits is:

$$p = \frac{t!}{(t-n)!t^n} = \prod_{i=0}^{n-1} \frac{(t-i)}{t} = \prod_{i=0}^{n-1} \left(1 - \frac{i}{t}\right)$$

Assuming the favourable case $n \ll t$ and using the approximation $e^x \approx (1+x)$ for $|x| \ll 1$, we obtain:

$$p \approx \prod_{i=0}^{n-1} e^{-i/t} = e^{\sum_{i=0}^{n-1} -i/t} = e^{-\frac{n(n-1)}{2t}} = e^{\frac{n-n^2}{2t}}$$

Since $n^2 \gg n$ for large n, a good approximation for p is given by:

$$p \approx e^{-\frac{n^2}{2t}}$$

The corresponding probability of state omission is $q = 1 - p$. Unfortunately the table sizes required to keep the probability of state omission very low are impractically large. For example, to obtain a state omission probability of 0.1% when inserting $n = 10^6$ states requires the allocation of a bit vector of 125TB. The situation can be improved a little by using two independent hash functions h_1 and h_2. When inserting a state s, both $T[h_1(s)]$ and $T[h_2(s)]$ are set to one. Likewise, we conclude s has been explored only if both $T[h_1(s)]$ and $T[h_2(s)]$ are set to one. Wolper and Leroy [35] show that now the probability of no hash collisions is:

$$p \approx e^{-\frac{4n^3}{t^2}}.$$

However the table sizes required to keep the probability of state omission low are still impractically large. Using more than two hash functions helps improve

the probability slightly; in fact it turns out that the optimal number of functions is about 20 [35]. However, computing 20 independent hash functions on every state is expensive and the resulting algorithm is very slow. The strength of Holzmann's algorithm therefore lies in the goal for which it was originally designed, i.e. the ability to maximize coverage in the face of limited memory, and not in its ability to provide complete state coverage.

Wolper and Leroy's Hash Compaction. Holzmann's method requires a very low ratio of states to hash table entries to provide a good probability of complete state space coverage. Consequently, a large amount of the space allocated to the bit vector will be wasted. Wolper and Leroy observed that it would be better to store which bit positions in the table are occupied instead [35]. This can be done by hashing states onto compressed keys of b bits. These keys can then be stored in a smaller hash table which supports a collision resolution scheme.

Given a hash table with $m \geq n$ slots, the memory required is:

$$M = (mb + m)/8 = m(b + 1)/8$$

since we need to store the keys, as well as a bit vector indicating which hash table slots are occupied. If we wish to construct the state graph efficiently, states also need to be assigned unique state sequence numbers. Given s-bit state sequence numbers, total memory consumption in this case is:

$$M = m(b + s + 1)/8.$$

In terms of the reliability of the technique, this approach is equivalent to a bit-state hashing scheme with a table size of 2^b, so the probability of no collision p is given by:

$$p \approx e^{-\frac{n^2}{2^{b+1}}}$$

Wolper and Leroy recommend compressed values of $b = 64$ bits, i.e. 8-byte compression.

Stern and Dill's Improved Hash Compaction. Wolper and Leroy do not discuss exactly how states are mapped onto slots in their hash table. It seems to be implicitly assumed that the hash values used to determine where to store the b-bit compressed values in the hash table are calculated using the b-bit compressed values themselves. Stern and Dill [36] noticed that the omission probability can be dramatically reduced in two ways – firstly by calculating the hash values and compressed values independently and secondly by using a collision resolution scheme which keeps the number of probes per insertion low. This improved technique is so effective that it requires only 5 bytes per state in situations where Wolper and Leroy's standard hash compaction requires 8 bytes per state.

Given a hash table with m slots, states are inserted into the table using two hash functions $h_1(s)$ and $h_2(s)$. These hash functions generate the probe sequence $h^{(0)}(s), h^{(1)}(s), \ldots, h^{(m-1)}(s)$ with $h^{(i)}(s) = (h_1(s) + ih_2(s)) \bmod m$

for $i = 0, 1, \ldots, m-1$. This double hashing scheme prevents the clustering associated with simple rehashing algorithms such as linear probing. A separate independent compression function h_3 is used to calculate the b-bit compressed state values which are stored in the table.

Slots are examined in the order of the probe sequence, until one of two conditions are met:

1. If the slot currently being examined is empty, the compressed value is inserted into the table at that slot.
2. If the slot is occupied by a compressed value equal to the $h_3(s)$, we assume (possibly incorrectly) that the state has already been explored.

Total memory consumption is the same as for Wolper and Leroy's hash compaction method, i.e.

$$M = m(b + s + 1)/8$$

where we assume a bit vector indicates which hash slots are used, and s-bit unique state sequence numbers are used to identify states for efficient construction of the state graph.

Given m slots in the hash table, n of which are occupied by states, Stern and Dill prove that the probability of no state omissions p is given by

$$p \approx \prod_{k=0}^{n-1} \left[\sum_{j=0}^{k} \left(\frac{2^b - 1}{2^b} \right)^j \frac{m-k}{m-j} \prod_{i=0}^{j-1} \frac{k-i}{m-i} \right]$$

This formula takes $O(n^3)$ operations to evaluate. Stern and Dill derive an $O(1)$ approximation given by

$$p \approx \left(\frac{2^b - 1}{2^b} \right)^{(m+1)\ln(\frac{m+1}{m-n+1}) - \frac{n}{2(m-n+1)} + \frac{2n+2mn-n^2}{12(m+1)(m-n+1)^2} - n}$$

An upper bound for the probability of state omission q is

$$q \leq \frac{1}{2^b} \left[(m+1)(H_{m+1} - H_{m-n+1}) - n \right]$$

where $H_n = \sum_{k=1}^{n} 1/k$ is the nth harmonic number [36]. This probability rises sharply as the hash table becomes full, since compressed states being inserted are compared against many compressed values before an empty slot is found. Stern and Dill derive a more straightforward formula for the approximate maximum omission probability for a full table (i.e. with $m = n$):

$$q \approx \frac{1}{2^b} m(\ln m - 1)$$

which shows the omission probability is approximately proportional to $m \ln m$. Increasing b, the number of bits per state, by one roughly halves the maximum omission probability.

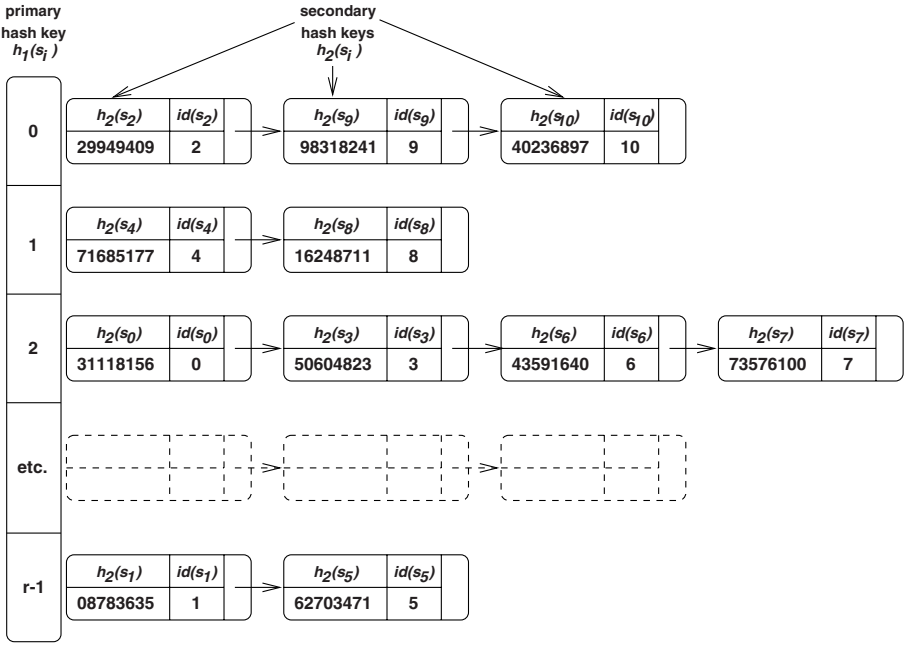


Fig. 2. Layout of the explored state table under the dynamic probabilistic hash compaction scheme

Dynamic Probabilistic State Space Generation. We now discuss a probabilistic technique which uses dynamic storage allocation and which yields a very low collision probability [37]. The system is illustrated in Fig. 2. Here, the explored state table takes the form of a hash table with several rows. Attached to each row is a linked list which stores compressed state descriptors and state sequence numbers.

Two independent hash functions are used. Given a state descriptor s, the *primary* hash function $h_1(s)$ is used to determine which hash table row should be used to store a compressed state, while the *secondary* hash function $h_2(s)$ is used to compute a compressed state descriptor value (also known as a secondary key). If a state's secondary key $h_2(s)$ is present in the hash table row given by its primary key $h_1(s)$, then the state is deemed to be the already-explored state identified by the sequence number $\mathrm{id}(s)$. Otherwise, the secondary key and a new sequence number are added to the hash table row and the state's successors are added onto the FIFO queue.

Fig. 3 shows the complete sequential dynamic probabilistic state space generation algorithm based on our hash compaction technique. Here H represents the state hash table in which each state $s \in E$ has an entry of form $[h_1(s), h_2(s)]$. Since it is now not necessary to store the full state space E in memory, the insertion of states into E can be handled by writing the states to a disk file as they are encountered.

```
begin
    H = {[h1(s0), h2(s0)]}
    F.add(s0)
    E = { s0 }
    A = ∅
    while (F not empty) do begin
        F.remove(s)
        for each s' ∈ succ(s) do begin
            if [h1(s'), h2(s')] ∉ H do begin
                F.add(s')
                E = E ∪ {s'}
                H = H ∪ {[h1(s'), h2(s')]}
            end
            A = A ∪ {id(s) → id(s')}
        end
    end
end
```

Fig. 3. Sequential dynamic probabilistic state space generation algorithm

Note that two states s_1 and s_2 are classified as being equal if and only if $h_1(s_1) = h_1(s_2)$ and $h_2(s_1) = h_2(s_2)$. This may happen even when the two states are different, so collisions may occur (as in all other probabilistic methods). However, as we will see below, the probability of such a collision can be kept very small – certainly much smaller than the chance of a serious man-made error in the specification of the model. In addition, by regenerating the state space with different sets of independent hash functions and comparing the resulting number of states and transitions, it is possible to further arbitrarily decrease the risk of an undetected collision.

We now calculate the probability of complete state coverage p. We consider a hash table with r rows and $t = 2^b$ possible secondary key values, where b is the number of bits used to store the secondary key. In such a hash table, there are rt possible ways of representing a state. Assuming that $h_1(s)$ and $h_2(s)$ distribute states randomly and independently, each of these representations are equally likely. Thus, if there are n distinct states to be inserted into the hash table, the probability p that all states are uniquely represented is given by:

$$p = \frac{(rt)!}{(rt-n)!(rt)^n} \tag{24}$$

An equivalent formulation of Eq. (24) is:

$$p = \prod_{i=0}^{n-1} \frac{rt-i}{rt} = \prod_{i=0}^{n-1} \left(1 - \frac{i}{rt}\right) \tag{25}$$

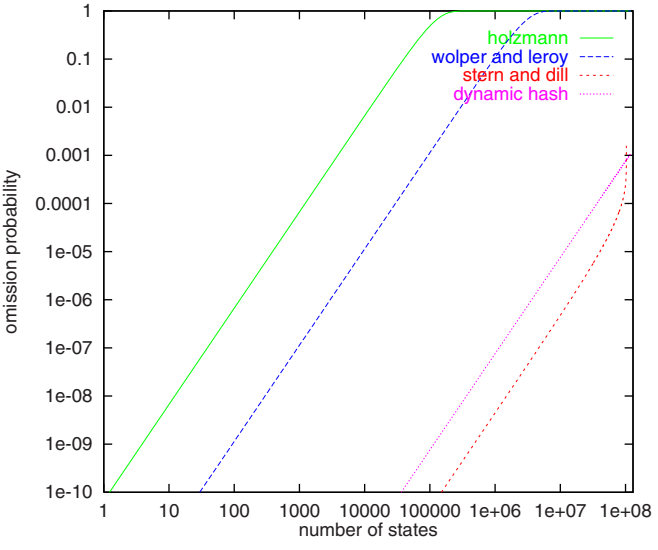


Fig. 4. Contemporary static probabilistic methods compared with the dynamic hash compaction method in terms of omission probability

Assuming $n \ll rt$ and using the fact that $e^x \approx (1+x)$ for $|x| \ll 1$, we obtain:

$$p \approx \prod_{i=0}^{n-1} e^{-i/rt} = e^{\sum_{i=0}^{n-1} -i/rt} = e^{-\frac{n(n-1)}{2rt}} = e^{\frac{n-n^2}{2rt}}$$

Since $n^2 \gg n$ for large n, a simple approximation for p is given by:

$$p \approx e^{-\frac{n^2}{2rt}} \tag{26}$$

It can be shown that if $n^2 \ll rt$ then this approximation is also a lower bound for p (and thus provides a conservative estimate for the probability of complete state coverage) [10].

The corresponding upper bound for the probability q that all states are not uniquely represented, resulting in the omission of one or more states from the state space, is of course simply:

$$q = 1 - p \leq \frac{n^2}{2rt} = \frac{n^2}{r2^{b+1}}. \tag{27}$$

Thus the probability of state omission q is proportional to n^2 and is inversely proportional to the hash table size r. Increasing the size of the compressed state descriptors b by one bit halves the omission probability.

Comparison of State Omission Probabilities. Fig. 4 compares the omission probability of contemporary static probabilistic methods with that of the

Table 1. Parameters used in the comparison of omission probabilities

Method	Parameters
Holzmann	$l = 7.488 \times 10^9$ bits
	$M = 91.4$ MB

Method	Parameters
Wolper and Leroy	$b = 42$ bits
	$s = 32$ bits
	$m = 10^8$ slots
	$M = 91.6$ MB

Method	Parameters
Stern and Dill	$b = 40$ bits
	$s = 32$ bits
	$m = 10.26 \times 10^8$ slots
	$M = 91.4$ MB

Method	Parameters
Dynamic hash	$b = 40$ bits
	$s = 32$ bits
	$r = 6\,000\,000$ rows
	$h = 6$ bytes
	$M = 91.4$ MB (for $n = 10^8$)

dynamic hash compaction method for state space sizes of various magnitudes up to 10^8. The parameters used for each method are presented in Tab. 1, and are selected such that the memory use of all four algorithms is the same. The graph shows that the dynamic method yields a far lower omission probability than both Holzmann's method and Wolper and Leroy's method. In addition, the dynamic method is competitive with Stern and Dill's algorithm and yields a better omission probability when the hash table becomes full or nearly full.

Parallel Dynamic Probabilistic State Space Generation. We now investigate how our technique can be enhanced to take advantage of the memory and processing power provided by a network of workstations or a distributed-memory parallel computer. We assume there are N nodes available and that each processor has its own local memory and can communicate with other nodes via a network.

In the parallel algorithm, the state space is partitioned between the nodes so that each node is responsible for exploring a portion of the state space and for constructing part of the state graph. A partitioning hash function $h_0(s) \rightarrow (0, \ldots, N-1)$ is used to assign states to nodes, such that node i is responsible for exploring the set of states E_i and for constructing the portion of the state graph A_i where:

$$E_i = \{s : h_0(s) = i\}$$
$$A_i = \{(s_1 \rightarrow s_2) : h_0(s_1) = i\}$$

It is important that $h_0(s)$ achieves a good spread of states across nodes in order to achieve good load balance. Naturally, the values produced by $h_0(s)$ should also be independent of those produced by $h_1(s)$ and $h_2(s)$ to enhance the reliability of the algorithm. Guidelines for choosing hash functions which meet these goals are discussed in [10].

The operation of node i in the parallel algorithm is shown in Fig. 5. Each node i has a local FIFO queue F_i used to hold unexplored local states and a

```
begin
    if h_0(s_0) = i do begin
        H_i = {[h_1(s_0), h_2(s_0)]}
        F_i.add(s_0)
        E_i = {s_0}
    end else
        H_i = E_i = ∅
    A_i = ∅
    while (shutdown signal not received) do begin
        if (F_i not empty) do begin
            s = F_i.remove()
            for each s' ∈ succ(s) do begin
                if h_0(s') = i do begin
                    if [h_1(s'), h_2(s')] ∉ H_i do begin
                        H_i = H_i ∪ {[h_1(s'), h_2(s')]}
                        F_i.add(s')
                        E_i = E_i ∪ {s'}
                    end
                    A_i = A_i ∪ {id(s) → id(s')}
                end else
                    send-state(h_0(s'), id(s), s')
            end
        end
        while (receive-id(g, h)) do
            A_i = A_i ∪ {g → h}
        while (receive-state(k, g, s')) do begin
            if [h_1(s'), h_2(s')] ∉ H_i do begin
                H_i = H_i ∪ {h_1(s'), h_2(s')}
                F_i.add(s')
                E_i = E_i ∪ {s'}
            end
            send-id(k, g, id(s'))
        end
    end
end
```

Fig. 5. Parallel state space generation algorithm for node i

hash table H_i representing a compressed version of the set E_i, i.e. those states which have been explored locally. State s is assigned to processor $h_0(s)$, which stores the state's compressed state descriptor $h_2(s)$ in the local hash table row given by $h_1(s)$. As before, it is not necessary to store the complete state space E_i in memory, since states can be written out to a disk file as they are encountered.

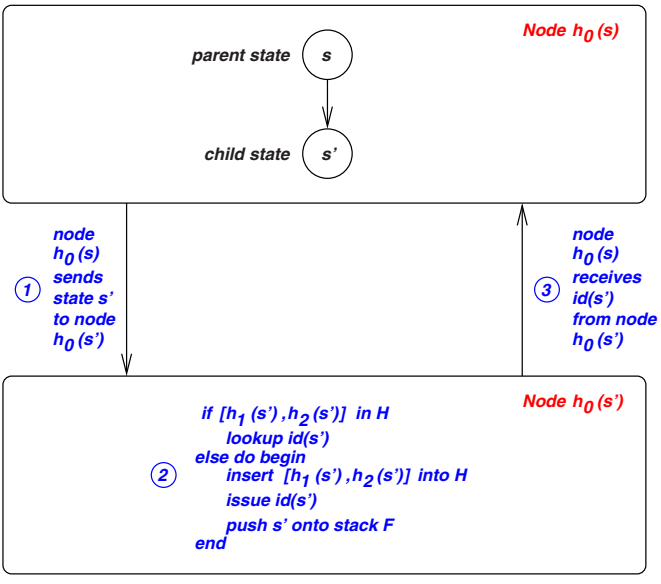


Fig. 6. Steps required to identify child state s' of parent s

Node i proceeds by removing a state from the local FIFO queue and determining the set of successor states. Successor states for which $h_0(s) = i$ are dealt with locally, while other successor states are sent to the relevant remote processors via calls to send-state(k, g, s). Here k is the remote node, g is the identity of the parent state and s is the state descriptor of the child state. The remote processors must receive incoming states via matching calls to receive-state(k, g, s) where k is the sender node. If they are not already present, the remote processor adds the incoming states to both the remote state hash table and FIFO queue.

For the purpose of constructing the state graph, states are identified by a pair of integers (i, j) where $i = h_0(s)$ is the node number of the host processor and j is the local state sequence number. As in the sequential case, the index j can be stored in the state hash table of node i. However, a node will not be aware of the state identity numbers of non-local successor states. Therefore, when a node receives a state it returns its identity to the sender by calling send-id(k, g, h) where k is the sender, g is the identity of the parent state and h is the identity of the received state. The identity is received by the original sender via a call to receive-id(g, h). Fig. 6 summarises the main steps that take place to identify and process each child s' of state s in the case that $h_0(s) \neq h_0(s')$.

In practice, it is inefficient to implement the communication as detailed in Fig. 5 and Fig. 6, since the network rapidly becomes overloaded with too many short messages. Consequently state and identity messages are buffered and sent in large blocks. In order to avoid starvation and deadlock, nodes that have very few states left in their FIFO queue or are idle broadcast a message to other nodes requesting them to flush their outgoing message buffers.

The algorithm terminates when all the F_i are empty and there are no outstanding state or identity messages. The problem of determining when these conditions are satisfied across a distributed set of processes is a non-trivial problem. From the several distributed termination algorithms surveyed in [38], we have chosen to use Dijkstra's circulating probe algorithm [39].

Reliability. Using the parallel algorithm, two distinct states s_1 and s_2 will be mistakenly classified as identical states if and only if $h_0(s_1) = h_0(s_2)$ and $h_1(s_1) = h_1(s_2)$ and $h_2(s_1) = h_2(s_2)$. Since h_0, h_1 and h_2 are independent functions, the reliability of the parallel algorithm is essentially the same as that of the sequential algorithm with a large hash table of Nr rows, giving a state omission probability of

$$q = \frac{n^2}{Nr2^{b+1}}. \tag{28}$$

Space Complexity In the parallel algorithm, each node supports a hash table with r rows. This requires a total of Nhr bytes of storage. The total amount of space required for the dynamic storage of n states remains the same as for the sequential version, i.e. $(b+s)n/8$ bytes. Thus the total memory requirement across all nodes is given by:

$$M = Nhr + n(b+s)/8.$$

4.2 Parallel Disk-Based Steady State Solution

Having generated the state space and state graph, the next challenge in performance analysis is usually to find the long run proportion of time the system spends in each of its states. The state graph maps directly onto a continuous time Markov chain which can then be solved for its steady-state distribution, according to Eq. (2).

Since the resources of a single workstation are usually inadequate to tackle the solution of large models (e.g. simply storing the solution vector of a system with 100 million states requires 800MB memory), we explore distributed out-of-core techniques which leverage the compute power, memory and disk space of several processors.

Scalable Numerical Methods. A broad spectrum of sequential solution techniques are available for solving steady-state equations [40]. These include classical iterative methods, Krylov subspace techniques and decomposition-based techniques. Many of these algorithms are unsuited to distributed or parallel implementation, however, since they rely on the so-called "Gauss-Seidel effect" to accelerate convergence. This effect occurs when newly updated steady-state vector elements are used in the calculation of other vector elements within the same iteration. In the case of sparse matrices, this sequential dependency can be alleviated by using multi-coloured ordering schemes which allow parallel computation of unrelated vector elements in phases; however, finding such orderings

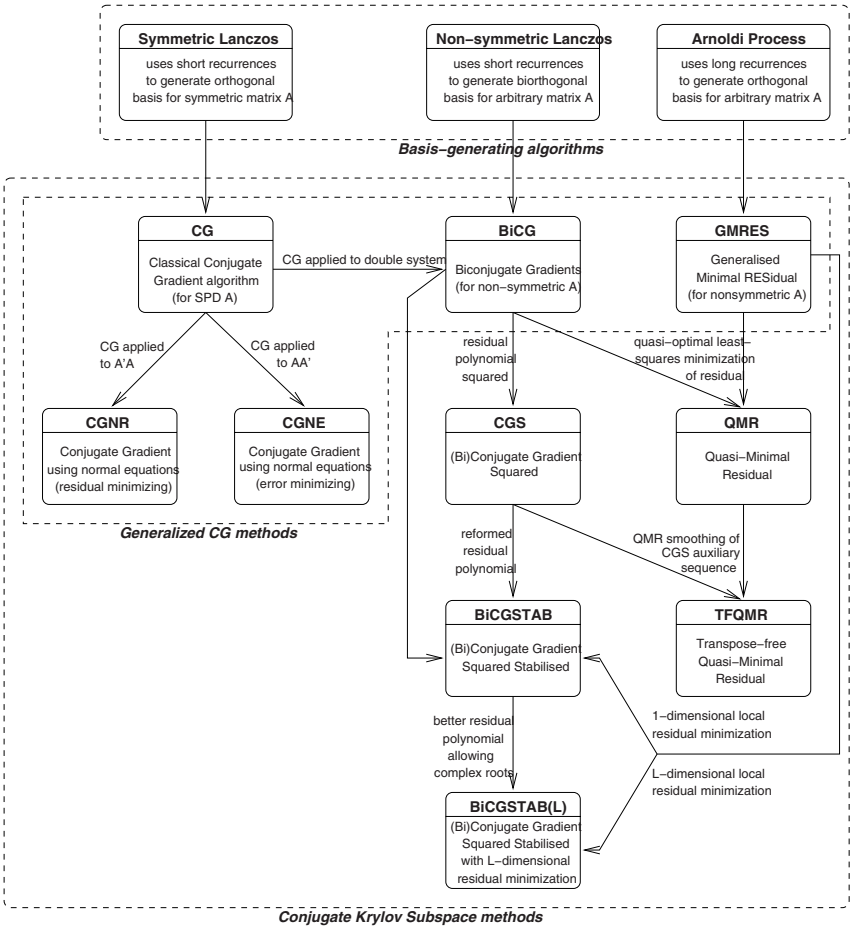


Fig. 7. An overview of Krylov subspace techniques

is a combinatorial problem of exponential complexity. Consequently obtaining suitable orderings for very large matrices is infeasible.

Most classical iterative methods, such as Gauss-Seidel and Successive Overrelaxation (SOR), suffer from this problem. An important exception is the Jacobi method which uses independent updates of vector elements. The Jacobi method is characterised by slow, smooth convergence.

Krylov subspace methods [41] are a powerful class of iterative methods which includes many conjugate gradient-type algorithms. They derive their name from the fact that they generate their iterates using a shifted Krylov subspace associated with the coefficient matrix. They are widely used in scientific computing since they are parameter free (unlike SOR) and exhibit rapid, if somewhat erratic, convergence. In addition, these methods are well suited to parallel implementation because they are based on matrix–vector products, independent

vector updates and inner products. Fig. 7 presents a conceptual overview of the most important techniques. The arrows show the relationships between the methods, i.e. how the methods have been generalised from their underlying basis-generating algorithms and also how key concepts have been inherited from one algorithm to the next.

The most recently developed Krylov subspace algorithms (such as CGS [42], BiCGSTAB [43] and TFQMR [44]) are also particularly suited to a disk-based implementation since they access $\mathbf{A}$ in a predictable fashion and do not require multiplication with $\mathbf{A}^T$. Compared to classical iterative methods, however, Krylov subspace techniques have high memory requirements. CGS is often used because it requires the least memory of these methods.

Disk-based Solution Techniques. The concept of using magnetic disk as a buffer to store data that is too large to fit into main memory is an idea which originated three decades ago with the development of overlays and virtual memory systems. However, only recently, with the widespread availability of large, cheap, high-bandwidth hard disks has attention been focused on the potential of disks as high-throughput data sources appropriate for use in data-intensive computations.

In [45] and [46], Deavours and Sanders make a compelling case for the potential of disk-based steady-state solution methods for large Markov models. They note that our ability to solve large matrices is limited by the memory required to store a representation of the transition matrix and by the effective rate at which matrix elements can be produced from the encoding. As a general rule, the more compact the representation, the more CPU overhead is involved in retrieving matrix elements. Two common encodings are Kronecker representations and "on-the-fly" methods. Deavours and Sanders estimate the effective data production rate of Kronecker and "on-the-fly" methods as being 2 MB/s and 440 KB/s respectively on their 120 MHz HP C110 workstation. Other published results show that an implementation of a state-of-the-art Kronecker technique running on a 450 MHz Pentium-II workstation yields an effective data production rate of around 2.5 MB/s [27].

At the same time, modern workstation disks are capable of sustaining data transfer rates in excess of 20 MB/s. This suggests that it would be worthwhile to store the transition matrix on disk, given that enough disk space is available and given that we can apply an iterative solution method that accesses the transition matrix in a predictable way. Such an approach has the potential to produce data faster than both Kronecker and on-the-fly methods, without any of the structural restrictions inherent in Kronecker methods.

Deavours and Sanders demonstrate the effectiveness of this approach by devising a sequential disk-based solution tool which makes use of two cooperating processes. One of the processes is dedicated to reading disk data while the other performs computation using a Block Gauss-Seidel algorithm, thus allowing for the overlap of disk I/O and computation. The processes communicate using semaphores and shared memory. The advantage of using Block Gauss-Seidel is

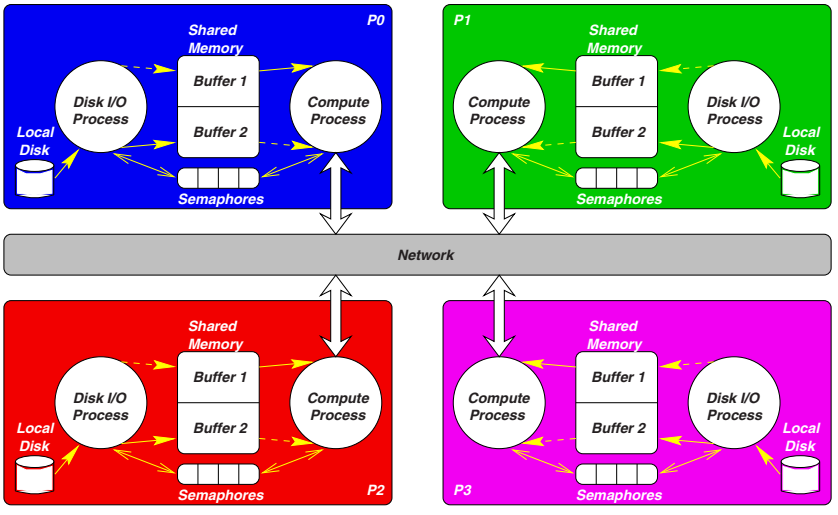


Fig. 8. Distributed disk-based solver architecture

that diagonal matrix blocks can be read from disk once, be cached in memory and then reused several times.

The memory required by the disk-based approach is small – besides the shared memory buffers, space is only required for the solution vector itself. This enables the solution of extremely large models with over 10 million states and 100 million non-zero entries on a HP C110 workstation with 128MB RAM and 4GB of disk space in just over 5 hours.

Kwiatkowska and Mehmood reduce the memory requirements of disk-based methods even further by proposing a block-based Gauss-Seidel method which uses disk to store blocks of the steady-state vector as well as matrix blocks [47,48]. In this way, a model of a manufacturing system with 133 million states is solved on a single PC in 13 days and 9 hours.

Parallel disk-based solver architectures have also been implemented with some success. Fig. 8 shows the architecture proposed in [49]. Each node has two processes: a *Disk I/O* process dedicated to reading matrix elements from a local disk, and a *Compute* process which performs the iterations using a Jacobi or CGS-based matrix–vector multiply kernel. The processes share two data buffers located in shared memory and synchronise via semaphores. Together the processes operate as a classical producer-consumer system, with the disk I/O process filling one shared memory buffer while the compute process consumes data from the other.

Bell and Haverkort apply a similar architecture in solving a 724 million state Markov chain model on a 26 node PC cluster in 16 days [50].

Hypergraph Partitioning. Any distributed solution scheme involves partitioning the sparse matrix and vector elements across the processors. Such

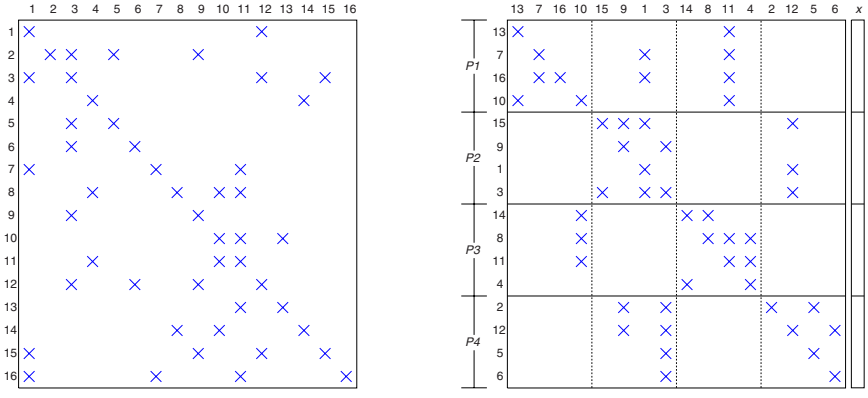


Fig. 9. A 16×16 non-symmetric sparse matrix (left), with corresponding 4-way hypergraph partition (right) and corresponding partitions of the vector

schemes necessitate the exchange of data (vector elements and possibly partial sums) after every iteration in the solution process. The objective in partitioning the matrix is to minimise the amount of data which needs to be exchanged while balancing the computational load (as given by the number of non-zero elements assigned to each processor).

Hypergraph partitioning is an extension of graph partitioning. Its primary application to date has been in VLSI circuit design, where the objective is to cluster pins of devices such that interconnect is minimised. It can also be applied to the problem of allocating the non-zero elements of sparse matrices across processors in parallel computation [51].

Formally, a hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{N})$ is defined by a set of vertices $\mathcal{V}$ and a set of nets (or hyperedges) $\mathcal{N}$, where each net is a subset of the vertex set $\mathcal{V}$ [51]. In the context of a row-wise decomposition of a sparse matrix $\mathbf{A}$, matrix row i $(1 \leq i \leq n)$ is represented by a vertex $v_i \in \mathcal{V}$ while column j $(1 \leq j \leq n)$ is represented by net $N_j \in \mathcal{N}$. The vertices contained within net N_j correspond to the row numbers of the non-zero elements within column j, i.e. $v_i \in N_j$ if and only if $a_{ij} \neq 0$. The weight of vertex i is given by the number of non-zero elements in row i, while the weight of a net is its contribution to the edge cut, which is defined as one less than the number of different partitions spanned by that net. The overall objective of a hypergraph sparse matrix partitioning is to minimise the sum of the weights of the cut nets while maintaining a balance criterion. A column-wise decomposition is achieved in an analogous fashion.

The matrix on the right of Fig. 9 shows the result of applying hypergraph-partitioning to the matrix on the left in a four-way row-wise decomposition. Although the number of off-diagonal non-zeros is 18 the number of vector elements which must be transmitted between processors during each matrix–vector multiplication (the communication cost) is 6. This is because the hypergraph partitioning algorithms not only aim to concentrate the non-zeros on the diagonals

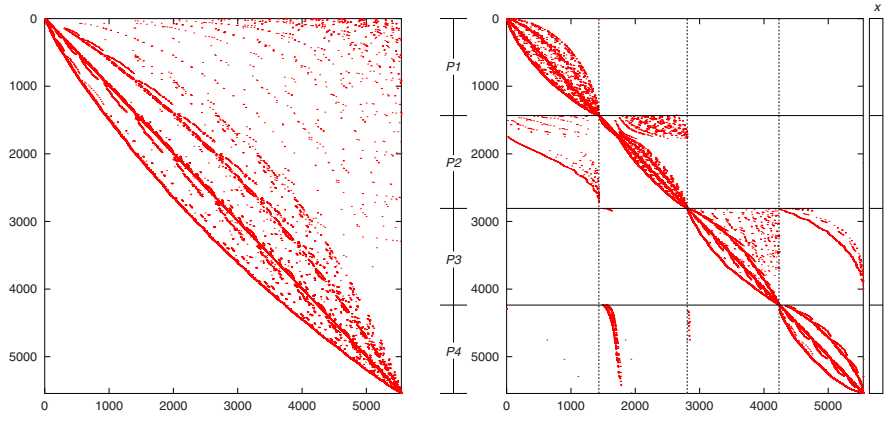


Fig. 10. Transposed transition matrix (left) and corresponding hypergraph-partitioned matrix (right)

but also strive to line up the off-diagonal non-zeros in columns. The edge cut of the decomposition is also 6, and so the hypergraph partitioning edge cut metric exactly quantifies the communication cost. This is a general property and one of the key advantages of using hypergraphs – in contrast to graph partitioning, where the edge cut metric merely approximates communication cost. Optimal hypergraph partitioning is NP-complete but there are a small number of hypergraph partitioning tools which implement fast heuristic algorithms, for example PaToH [51], hMeTiS [52] and Par*k*way [53].

Table 2. Communication overhead (left) and interprocessor communication matrix (right)

processor	non-zeros	local %	remote %	reused %
1	7 022	99.96	0.04	0
2	7 304	91.41	8.59	34.93
3	6 802	88.44	11.56	42.11
4	6 967	89.01	10.99	74.28

	1	2	3	4
1	-	407	-	4
2	3	-	16	181
3	-	-	-	12
4	-	1	439	-

Fig. 10 shows the application of hypergraph partitioning to a (transposed) generator matrix. Statistics about the communication associated with this decomposition for a single matrix–vector multiplication are presented in Tab. 2. We see that around 90% of the non-zero elements allocated to each processor are local, i.e. they are multiplied with vector elements that are stored locally. The remote non-zero elements are multiplied with vector elements that are sent from other processors. However, because the hypergraph decomposition tends to align remote non-zero elements in columns (well illustrated in the 2nd block

belonging to processor 4), reuse of received vector elements is good (up to 74%) with correspondingly lower communication overhead. The communication matrix on the right in Tab. 2 shows the number of vector elements sent between each pair of processors during each iteration (e.g. 181 vector elements are sent from processor 2 to processor 4).

4.3 Parallel Computation of Densities and Quantiles of First Passage Time

A rapid response time is an important performance criterion for almost all computer-communication and transaction processing systems. Response time quantiles are frequently specified as key quality of service metrics in Service Level Agreements and industry standard benchmarks such as TPC. Examples of systems with stringent response time requirements include mobile communication systems, stock market trading systems, web servers, database servers, flexible manufacturing systems, communication protocols and communication networks. Typically, response time targets are specified in terms of quantiles – for example "95% of all text messages must be delivered within 3 seconds".

In the past, numerical computation of analytical response time densities has proved prohibitively expensive except in some Markovian systems with restricted structure such as overtake-free queueing networks [54]. However, with the advent of high-performance parallel computing and the widespread availability of PC clusters, direct numerical analysis on Markov and semi-Markov chains has now become a practical proposition.

There are two main methods for computing first passage time (and hence response time) densities in Markov chains: those based on Laplace transforms and their inversion [55,56] and those based on uniformisation [8,6]. The former has wider application to semi-Markov processes but is less efficient than uniformisation when restricted to Markov chains.

Numerical Laplace Transform Inversion. The key to practical analysis of semi-Markov processes lies in the efficient representation of their general distributions. Without care the structural complexity of the SMP can be recreated within the representation of the distribution functions.

Many techniques have been used for representing arbitrary distributions – two of the most popular being *phase-type distributions* and *vector-of-moments* methods. These methods suffer from, respectively, exploding representation size under composition, and containing insufficient information to produce accurate answers after large amounts of composition.

As all our distribution manipulations take place in Laplace-space, we link our distribution representation to the Laplace inversion technique that we ultimately use. Our tool supports two Laplace transform inversion algorithms, which are briefly outlined below: the Euler technique [57] and the Laguerre method [58] with modifications summarised in [59].

Both algorithms work on the same general principle of sampling the transform function $L(s)$ at n points, $s_1, s_2, \ldots, s_n$ and generating values of $f(t)$ at m

user-specified t-points $t_1, t_2, \ldots, t_m$. In the Euler inversion case $n = km$, where k can vary between 15 and 50, depending on the accuracy of the inversion required. In the modified Laguerre case, $n = 400$ and, crucially, is independent of m.

The process of selecting a Laplace transform inversion algorithm is discussed later; however, whichever is chosen, it is important to note that calculating $s_i, 1 \leq i \leq n$ and storing all our distribution transform functions, sampled at these points, will be sufficient to provide a complete inversion. Key to this is that fact that matrix element operations, of the type performed in Eq. (39), (i.e. convolution and weighted sum) do not require any adjustment to the array of domain s-points required. In the case of a convolution, for instance, if $L_1(s)$ and $L_2(s)$ are stored in the form $\{(s_i, L_j(s_i)) \ : \ 1 \leq i \leq n\}$, for $j = 1, 2$, then the convolution, $L_1(s)L_2(s)$, can be stored using the same size array and using the same list of domain s-values, $\{(s_i, L_1(s_i)L_2(s_i)) \ : \ 1 \leq i \leq n\}$.

Storing our distribution functions in this way has three main advantages. Firstly, the function has constant storage space, independent of the distribution-type. Secondly, each distribution has, therefore, the same constant storage requirement even after composition with other distributions. Finally, the function has sufficient information about a distribution to determine the required passage time (and no more).

Summary of Euler Inversion. The Euler method is based on the Bromwich contour inversion integral, expressing the function $f(t)$ in terms of its Laplace transform $L(s)$. Making the contour a vertical line $s = a$ such that $L(s)$ has no singularities on or to the right of it gives:

$$f(t) = \frac{2e^{at}}{\pi} \int_0^\infty Re(L(a + iu)) \cos(ut)\, du \tag{29}$$

This integral can be numerically evaluated using the trapezoidal rule with step-size $h = \pi/2t$ and $a = A/2t$ (where A is a constant that controls the discretisation error), which results in the nearly alternating series:

$$f(t) \approx f_h(t) = \frac{e^{A/2}}{2t} Re(L(A/2t)) + \frac{e^{A/2}}{2t} \sum_{k=1}^{\infty} (-1)^k Re\left(L\left(\frac{A + 2k\pi i}{2t}\right)\right) \tag{30}$$

Euler summation is employed to accelerate the convergence of the alternating series infinite sum, so we calculate the sum of the first n terms explicitly and use Euler summation to calculate the next m. To give an accuracy of 10^{-8} we set $A = 19.1, n = 20$ and $m = 12$ (compared with $A = 19.1$, $n = 15$ and $m = 11$ in [57]).

Summary of Laguerre Inversion. The Laguerre method [58] makes use of the Laguerre series representation:

$$f(t) = \sum_{n=0}^{\infty} q_n l_n(t) \qquad : t \geq 0 \tag{31}$$

where the Laguerre polynomials l_n are given by:

$$l_n(t) = \left(\frac{2n-1-t}{n}\right) l_{n-1}(t) - \left(\frac{n-1}{n}\right) l_{n-2}(t) \tag{32}$$

starting with $l_0 = e^{t/2}$ and $l_1 = (1-t)e^{t/2}$, and:

$$q_n = \frac{1}{2\pi r^n} \int_0^{\pi} Q(re^{iu}) e^{-iru}\, \mathrm{d}u \tag{33}$$

where $r = (0.1)^{4/n}$ and $Q(z) = (1-z)^{-1} L((1+z)/2(1-z))$.

The integral in the calculation of q_n can be approximated numerically by the trapezoidal rule, giving:

$$q_n \approx \bar{q}_n = \frac{1}{2nr^n} \left(Q(r) + (-1)^n Q(-r) + 2 \sum_{j=1}^{n-1} (-1)^j Re\left(Q(re^{\pi ji/n})\right) \right) \tag{34}$$

As described in [59], the Laguerre method can be modified by noting that the Laguerre coefficients q_n are independent of t. This means that if the number of trapezoids used in the evaluation of q_n is fixed to be the same for every q_n (rather than depending on the value of n), values of $Q(z)$ (and hence $L(s)$) can be reused after they have been computed. Typically, we set $n = 200$. In order to achieve this, however, the scaling method described in [58] must be used to ensure that the Laguerre coefficients have decayed to (near) 0 by $n = 200$. If this can be accomplished, the inversion of a passage time density for any number of t-values can be achieved at the fixed cost of calculating 400 truncated summations of the type shown in Eq. (39). This is in contrast to the Euler method, where the number of truncated summations required is a function of the number of points at which the value of $f(t)$ is required.

Iterative Passage-Time Analysis for SMPs. Passage-time analysis in semi-Markov processes involves the solution of a set of linear equations in complex variables. In [60], we set out an efficient iterative approach to passage time calculation and proved its convergence to the analytic passage time distribution. The algorithm has since been implemented and is used to calculate semi-Markov passage times in the SMARTA tool (described below).

Recall the semi-Markov process, $Z(t)$, of Sect. 2.2, where $N(t)$ is the number of state transitions that have taken place by time t. We formally define the rth transition first passage time to be:

$$P_{i\boldsymbol{j}}^{(r)} = \inf\{u > 0 : Z(u) \in \boldsymbol{j}, N(u) > 0 \mid N(u) \leq r, Z(0) = i\} \tag{35}$$

which is the time taken to enter a state in $\boldsymbol{j}$ for the first time having started in state i at time 0 and having undergone up to r state transitions[1]. $P_{i\boldsymbol{j}}^{(r)}$ is a

[1] If there are immediate transitions in the semi-Markov process then we have to use a modified formulation of the passage time and iterative passage time definitions [60].

random variable with associated Laplace transform, $L^{(r)}_{ij}(s)$. $L^{(r)}_{ij}(s)$ is, in turn, the ith component of the vector:

$$\mathbf{L}^{(r)}_{\boldsymbol{j}}(s) = (L^{(r)}_{1\boldsymbol{j}}(s), L^{(r)}_{2\boldsymbol{j}}(s), \ldots, L^{(r)}_{N\boldsymbol{j}}(s)) \tag{36}$$

representing the passage time for terminating in $\boldsymbol{j}$ for each possible start state. This vector may be computed as:

$$\mathbf{L}^{(r)}_{\boldsymbol{j}}(s) = \mathbf{U}(\mathbf{I} + \mathbf{U}' + \mathbf{U}'^2 + \cdots + \mathbf{U}'^{(r-1)})\,\mathbf{e}_{\boldsymbol{j}} \tag{37}$$

where $\mathbf{U}$ is a matrix with elements $u_{pq} = r^*_{pq}(s)$ and $\mathbf{U}'$ is a modified version of $\mathbf{U}$ with elements $u'_{pq} = \delta_{p \notin \boldsymbol{j}}\, u_{pq}$, where states in $\boldsymbol{j}$ have been made absorbing. We include the initial $\mathbf{U}$-transition in Eq. (37), so as to generate cycle times for cases such as $L^{(r)}_{ii}(s)$ which would otherwise register as 0 if $\mathbf{U}'$ were used instead. The column vector $\mathbf{e}_{\boldsymbol{j}}$ has entries $e_{k\boldsymbol{j}} = \delta_{k \in \boldsymbol{j}}$.

From Eq. (15) and Eq. (35):

$$P_{i\boldsymbol{j}} = P^{(\infty)}_{i\boldsymbol{j}} \quad \text{and thus} \quad L_{i\boldsymbol{j}}(s) = L^{(\infty)}_{i\boldsymbol{j}}(s) \tag{38}$$

We can generalise to multiple source states $\boldsymbol{i}$ using the normalised steady-state vector $\boldsymbol{\alpha}$ of Eq. (21):

$$\begin{aligned} L^{(r)}_{\boldsymbol{ij}}(s) &= \boldsymbol{\alpha}\mathbf{L}^{(r)}_{\boldsymbol{j}}(s) \\ &= \textstyle\sum_{k=0}^{r-1} \boldsymbol{\alpha}\mathbf{U}\mathbf{U}'^k\,\mathbf{e}_{\boldsymbol{j}} \end{aligned} \tag{39}$$

The sum of Eq. (39) can be computed efficiently using sparse matrix–vector multiplications with a vector accumulator, $\boldsymbol{\mu}_r = \sum_{k=0}^{r} \boldsymbol{\alpha}\mathbf{U}\mathbf{U}'^k$. At each step, the accumulator (initialised as $\boldsymbol{\mu}_0 = \boldsymbol{\alpha}\mathbf{U}$) is updated with $\boldsymbol{\mu}_{r+1} = \boldsymbol{\alpha}\mathbf{U} + \boldsymbol{\mu}_r\mathbf{U}'$. The worst-case time complexity for this sum is $O(N^2 r)$ versus the $O(N^3)$ of typical matrix inversion techniques. In practice, for a sparse matrix with constant bandwidth (number of non-zeros per row), this can be as low as $O(Nr)$.

The SMARTA Tool. Our iterative passage-time analysis algorithm has been implemented in the SMARTA tool [61], the architecture of which is shown in Fig. 11. The process of calculating a passage time density begins with a high-level model specified in an enhanced form of the DNAmaca interface language [62,10]. This language supports the specification of stochastic Petri nets, queueing networks and stochastic process algebras. Next, a probabilistic, hash-based state generator [37] uses the high-level model description to produce the transition probability matrix P of the model's embedded Markov chain, the matrices $\mathbf{U}$ and $\mathbf{U}'$, and a list of the initial and target states. Normalised weights for the initial states are determined by the solution of $\boldsymbol{\pi} = \boldsymbol{\pi}\mathbf{P}$, which is readily done using any of a variety of steady-state solution techniques (e.g. [45,49]). $\mathbf{U}'$ is then partitioned using a hypergraph partitioning tool (a single partitioning is sufficient since all linear systems to be solved have the same non-zero structure).

Control is then passed to the distributed passage time density calculator, which is implemented in C++ using the Message Passing Interface (MPI) [63]

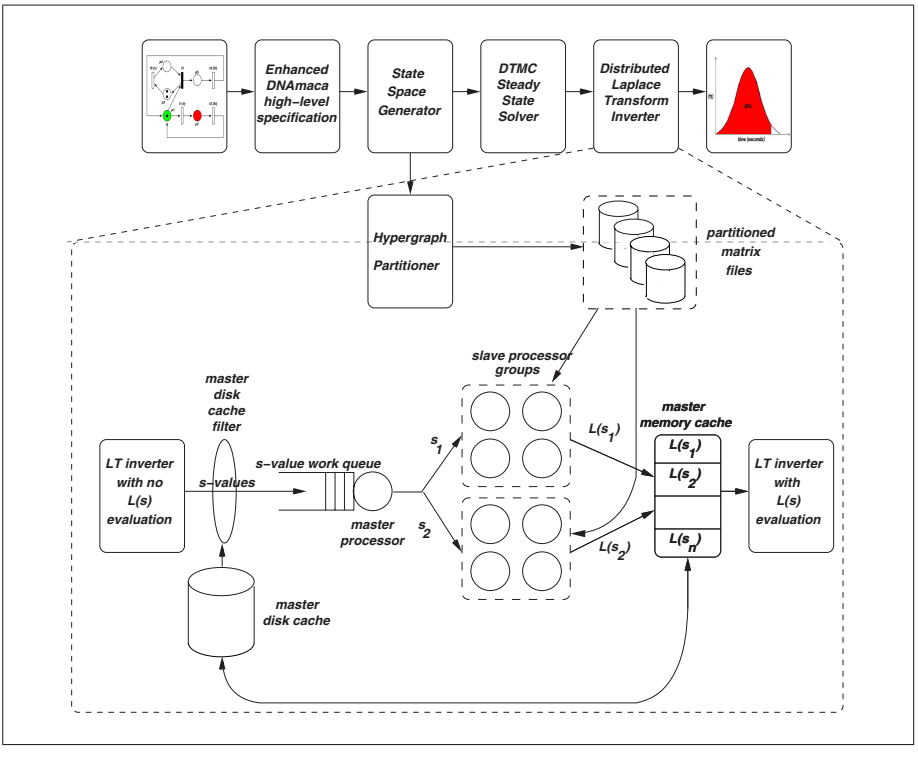


Fig. 11. Parallel hypergraph-based passage time density calculation pipeline

standard. This employs a master-slave architecture with groups of slave processors. The master processor computes in advance the values of s at which it will need to know the value of $L_{ij}(s)$ in order to perform the inversion. This can be done irrespective of the inversion algorithm employed. The s-values are then placed in a global work-queue to which the groups of slave processors make requests.

The highest ranking processor in a group of slaves makes a request to the master for an s-value and is assigned the next one available. This is then broadcast to the other members of the slave group to allow them to construct their columns of the matrix $\mathbf{U}'$ for that specific s. Each processor reads in the columns of the matrix $\mathbf{U}'$ that correspond to its allocated partition into two types of sparse matrix data structure and also reads in the initial source-state weighting vector $\boldsymbol{\alpha}$. *Local* non-zero elements (i.e. those elements in diagonal matrix blocks that will be multiplied with vector elements stored locally) are stored in a conventional compressed sparse column format. *Remote* non-zero elements (i.e. those elements in off-diagonal matrix blocks that must be multiplied with vector elements received from other processors) are stored in an ultrasparse matrix data structure – one for each remote processor – using a coordinate format. Each

processor then determines which vector elements need to be received from and sent to every other processor in the group on each iteration, adjusting the row indices in the ultrasparse matrices so that they index into a vector of received elements. This ensures that a minimum amount of communication takes place and makes multiplication of off-diagonal blocks with received vector elements efficient.

For each step in our iterative algorithm, each processor begins by using non-blocking communication primitives to send and receive remote vector elements, while calculating the product of local matrix elements with locally stored vector elements. The use of non-blocking operations allows computation and communication to proceed concurrently on parallel machines where dedicated network hardware supports this effectively. The processor then waits for the completion of non-blocking operations (if they have not already completed) before multiplying received remote vector elements with the relevant ultrasparse matrices and adding their contributions to the local vector-matrix product cumulatively.

Once the calculations of a slave group are deemed to have converged, the result is returned to the master by the highest-ranking processor in the group and cached. When all results have been computed and returned for all required values of s, the final Laplace inversion calculations are made by the master, resulting in the required t-points.

5 Application Examples

We demonstrate our analysis techniques on three application examples: a GSPN model of a communications protocol, a SM-SPN model of a voting system and a PEPA model of an active badge system.

5.1 Courier Protocol Model

Description. The GSPN shown in Fig. 12 (originally presented in [64]) models the ISO Application, Session and Transport layers of the Courier sliding-window communication protocol. Data flows from a sender ($p1$ to $p26$) to a receiver ($p27$ to $p46$) via a network. The sender's transport layer fragments outgoing data packets; this is modelled as two paths between $p13$ and $p35$. The path via $t8$ carries all fragments before the last one through the network to $p33$. Acknowledgements for these fragments are sent back to the sender (as signalled by the arrival of a token on $p20$), but no data is delivered to the higher layers on the receiver side. The path via $t9$ carries the last fragment of each message block. Acknowledgements for these fragments are generated and a data token is delivered to higher receiver layers via $t27$.

The average number of data packets sent is determined by the ratio of the weights on the immediate transitions $t8$ and $t9$. This ratio, known as the fragmentation ratio, is given by $q1 : q2$ (where $q1$ and $q2$ are the weights associated with transitions $t8$ and $t9$ respectively). Thus, this number of data packets is geometrically distributed, with parameter $q1/(q1+q2)$. Here we use a fragmentation ratio of one.

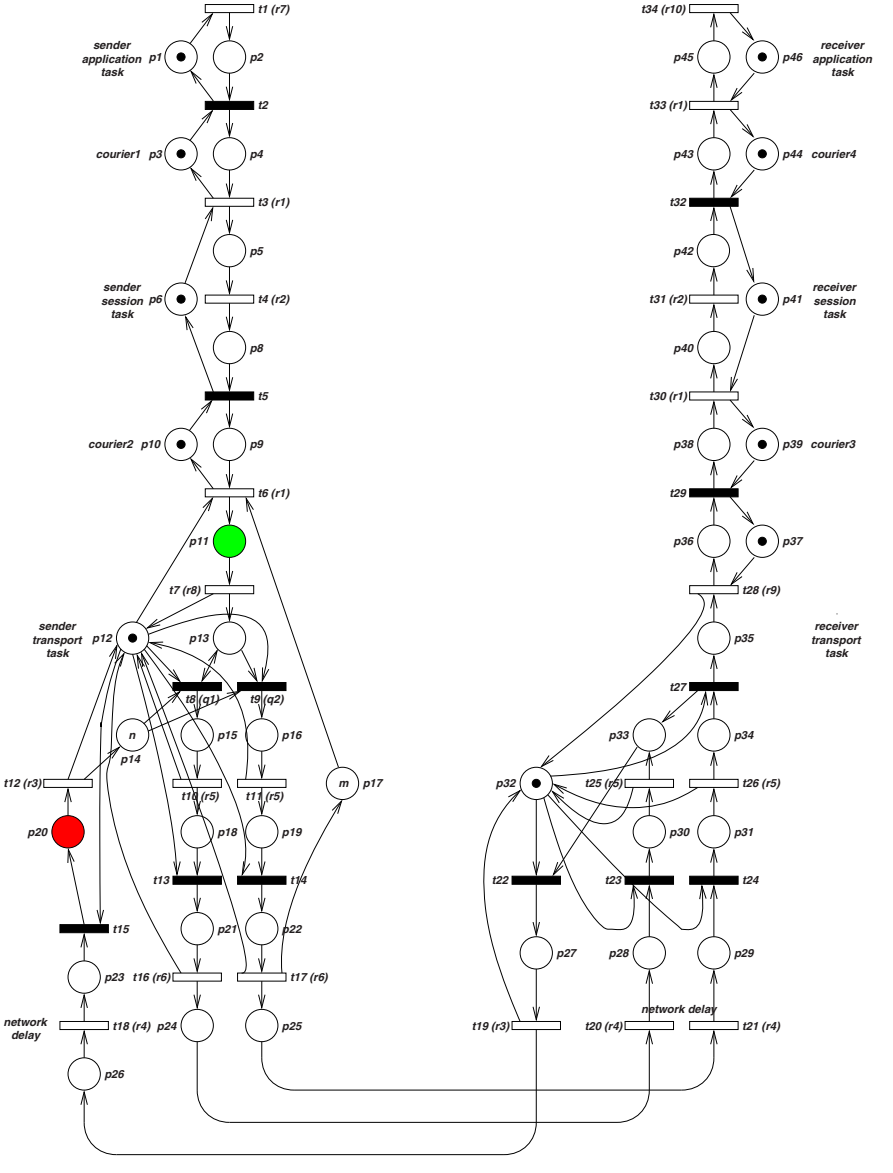


Fig. 12. The Courier Protocol Software Generalised Stochastic Petri net [64]

The transport layer is further characterised by two important parameters: the sliding window size n ($p14$) and the transport space m ($p17$). Different values of m and n yield state spaces of various sizes. The transition rates $r1, r2, \ldots, r10$ have the same relative magnitudes as those obtained by benchmarking a working implementation of the protocol (see [64]).

k	n	a
1	11 700	48 330
2	84 600	410 160
3	419 400	2 281 620
4	1 632 600	9 732 330
5	5 358 600	34 424 280
6	15 410 250	105 345 900
7	39 836 700	286 938 630
8	94 322 250	710 223 930

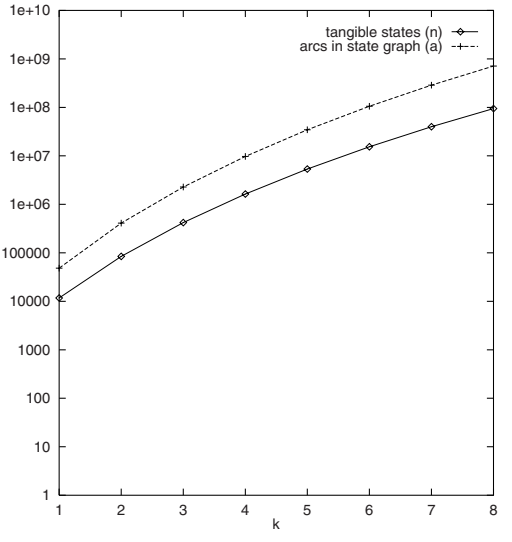


Fig. 13. The number of tangible states (n) and the number of arcs (a) in the state graph of the Courier model for various values of k

State Space Generation. We have implemented the state generation algorithm of Fig. 5 on a Fujitsu AP3000 distributed memory parallel computer. Our implementation is written in C++ with support for two popular parallel programming interfaces, viz. the Message Passing Interface (MPI) [65] and the Parallel Virtual Machine (PVM) interface [66]. The generator uses hash tables with $r = 750\,019$ rows per processor and $b = 40$ bit secondary keys. The results were collected using up to 16 processors on the AP3000. Each processor has a 300MHz UltraSPARC processor, 256MB RAM and a 4GB local disk. The nodes run the Solaris operating system and support MPI. They are connected by a high-speed wormhole-routed network with a peak throughput of 65MB/s.

The Courier model features a scaling parameter k (corresponding to the sliding window size) which we will vary to produce state graphs of different sizes (see Fig. 13).

The graph on the left of Fig. 14 shows the distributed run-time taken to explore Courier state spaces of various sizes (up to $k = 6$) using 1, 2, 4, 8, 12 and 16 processors on the AP3000. Each observed value is calculated as the mean of four runs. The $k = 5$ state space (5 358 600 states) can be generated on a single processor in 16 minutes 20 seconds; 16 processors require only 89 seconds. The $k = 6$ state space (15 410 250 states) can be generated on a single processor in 51 minutes 45 seconds; 16 processors require just 267 seconds.

The corresponding speedups for the cases $k = 1, 2, 3, 4, 5, 6$ are shown in the graph on the right of Fig. 14. For $k = 6$ using 16 processors, we observe a speedup of 11.65, giving an efficiency of 73%.

Memory utilisation is low – a single processor generating the $k = 5$ state space uses a total of 91MB (17.4 bytes per state), while the $k = 6$ state space requires

Table 3. Real time in seconds required for the distributed solution of the Courier model

		k=1	k=2	k=3	k=4	k=5	k=6	k=7	k=8
$p = 1$	Jacobi time (s)	33.647	278.18	1506.4	5550.3				
	Jacobi iterations	4925	4380	4060	3655				
	CGS time (s)	2.1994	21.622	163.87	934.27	29134			
	CGS iterations	60	81	106	129	157			
	Memory/node (MB)	20.3	22.1	30.5	60.8	154.0			
$p = 2$	Jacobi time (s)	29.655	176.62	1105.7	4313.6				
	Jacobi iterations	4925	4380	4060	3655				
	CGS time (s)	1.6816	13.119	93.28	509.90	7936.9			
	CGS iterations	57	84	107	131	148			
	Memory/node (MB)	20.2	21.1	25.45	41.2	89.7			
$p = 4$	Jacobi time (s)	25.294	148.45	627.96	3328.3				
	Jacobi iterations	4925	4380	4060	3655				
	CGS time (s)	1.2647	8.4109	58.302	322.50	1480.5			
	CGS iterations	60	80	108	133	159			
	Memory/node (MB)	20.1	20.6	22.9	31.4	57.5			
$p = 8$	Jacobi time (s)	38.958	140.06	477.02	1780.9	6585.4			
	Jacobi iterations	4925	4380	4060	3655	3235			
	CGS time (s)	1.4074	6.0976	39.999	204.46	934.76	4258.7		
	CGS iterations	61	82	109	132	155	171		
	Memory/node (MB)	20.0	20.3	21.7	26.5	41.4	81.6		
$p = 12$	Jacobi time (s)	32.152	133.58	457.23	1559.0	6329.2	11578	72202	
	Jacobi iterations	4925	4380	4060	3655	3235	2325	2190	
	CGS time (s)	1.4973	5.9345	34.001	157.73	852.53	2579.6	21220	
	CGS iterations	58	83	104	129	156	189	180	
	Memory/node (MB)	20.0	20.3	21.3	24.9	36.1	66.2	99.7	
$p = 16$	Jacobi time (s)	41.831	125.68	506.31	1547.9	5703.4	11683	32329	
	Jacobi iterations	4925	4380	4060	3650	3235	2325	2190	
	CGS time (s)	3.3505	7.1101	31.322	134.48	577.68	2032.5	13786	141383
	CGS iterations	60	91	104	132	146	173	179	213
	Memory/node (MB)	20.0	20.2	21.0	24.1	33.4	58.5	79.8	161

175MB (11.6 bytes per state). This is far less than the 94 bytes per state (45 16-bit integers plus a 32-bit unique state indentifier) that would be required by a straightforward exhaustive implementation.

Moving beyond the maximum state space size that can be generated on a single processor, on 16 processors we find that the $k = 7$ state space can be generated in around 10 minutes while just under 26 minutes are required to generate the $k = 8$ state space (with 94 322 250 states and 710 223 930 arcs).

Steady-state Analysis. Tab. 3 presents the execution time (defined as maximum processor run-time) in seconds required for the distributed disk-based solution of models using the CGS and Jacobi methods. The models range in size from $k = 1$ (11 700 states) to $k = 7$ (39.8 million states) and runs are conducted

Table 4. Courier Protocol performance measures in terms of the transport window size k

	$k=1$	$k=2$	$k=3$	$k=4$	$k=5$	$k=6$	$k=7$	$k=8$
λ	74.3467	120.372	150.794	172.011	187.413	198.919	207.690	214.477
P_{send}	0.01011	0.01637	0.02051	0.02334	0.02549	0.02705	0.02825	0.02917
P_{recv}	0.98141	0.96991	0.96230	0.95700	0.95315	0.95027	0.94808	0.94638
P_{sess1}	0.00848	0.01372	0.01719	0.01961	0.02137	0.02268	0.02368	0.02445
P_{sess2}	0.92610	0.88029	0.84998	0.82883	0.81345	0.80196	0.79320	0.78642
$P_{transp1}$	0.78558	0.65285	0.56511	0.50392	0.45950	0.42632	0.40102	0.38145
$P_{transp2}$	0.78871	0.65790	0.57138	0.51084	0.46673	0.43365	0.40835	0.38871

on 1, 2, 4, 8, 12 and 16 processors. The number of iterations for convergence and memory use per processor are also shown.

Fig. 15 compares the convergence of the Jacobi method with that of the CGS algorithm for the $k = 4$ case in terms of the number of matrix multiplications performed. As is typical for many models, the Jacobi method begins by converging quickly, but then plateaus, converging very slowly but smoothly. The CGS algorithm, on the other hand, exhibits erratic rapid convergence that improves in a concave fashion.

The largest state space solved is the $k = 8$ case (94 million states) which takes 1 day 15 hours of processing time on 16 processors. The total amount of I/O across all nodes is 3.4TB, with the nodes jointly processing an average of 24MB disk data every second.

Using the steady state vector, it is straightforward to derive some simple resource-based performance measures, as shown in Tab. 4. The most important is λ, the data throughput rate, which is given by the throughput of transition $t21$. Other measures yield task utilizations. In particular, $P_{transp1} = Pr\{p12 \text{ is marked}\} = Pr\{\text{transport task 1 is idle}\}$. Similarly we define $P_{transp2}$ for $p32$, P_{sess1} and P_{sess2} using $p6$ and $p41$, and P_{send} and P_{recv} using $p1$ and $p46$.

First Passage Time Analysis. We now apply our iterative passage-time analysis technique to determine the end-to-end response time from the initiation of a transport layer transmission to the arrival of the corresponding acknowledgement packet. Consequently we choose as source markings those markings for which $M(p_{11}) > 0$, and as destination markings those markings for which $M(p_{20}) > 0$. This approach works easily for a sliding window size of $n = 1$ since there can be only one outstanding unacknowledged packet. Naturally, if we wished to calculate the response time for sliding window sizes greater than one, we would need to augment the state vector used to describe markings to track the progress of a particular token through the Petri net.

The underlying reachability graph contains 29 010 markings, 11 700 of which are tangible and 17 310 of which are vanishing. There are 7 320 source markings and 1 680 destination markings. Fig. 16 shows the resulting numerical response time density. The median (50% quantile) and 95% quantile transmission times

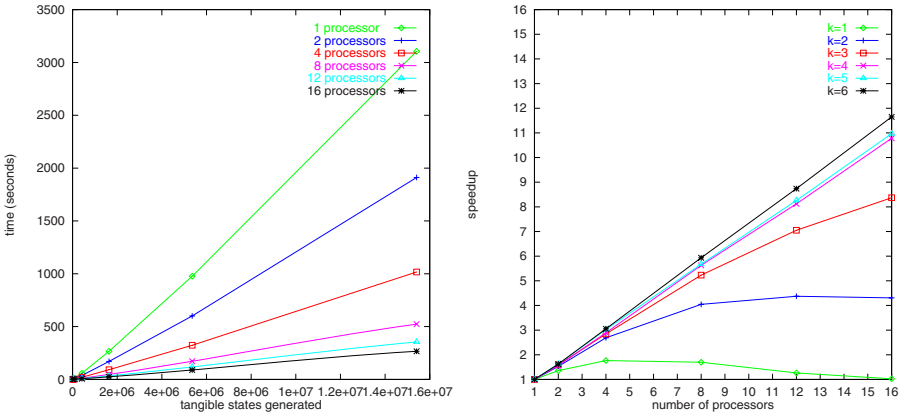


Fig. 14. Real time taken to generate Courier state spaces up to $k = 6$ using the original algorithm on 1, 2, 4, 8, 12 and 16 processors (left), and the resulting speedups for $k = 1, 2, 3, 4, 5$ and 6 (right)

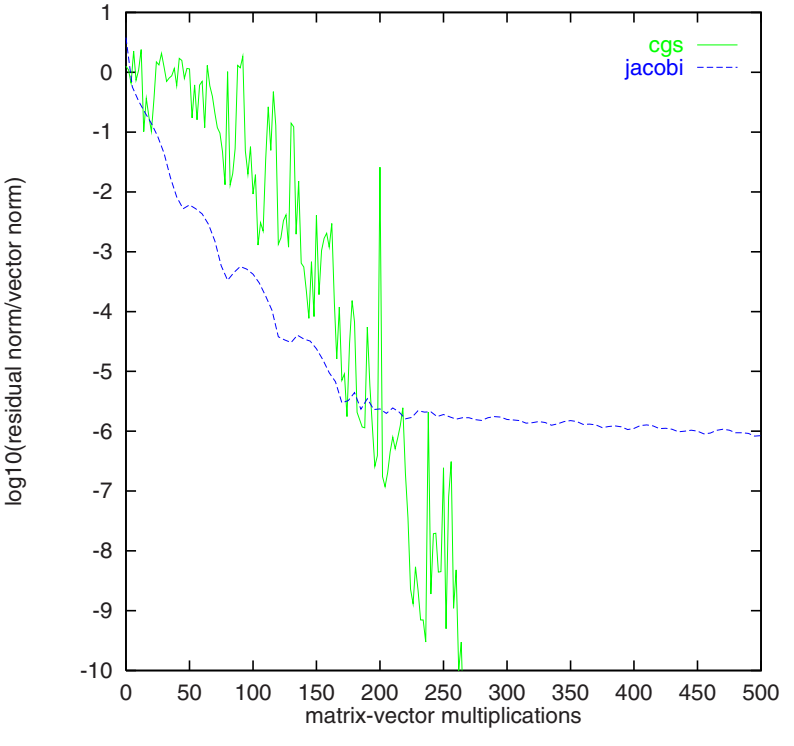


Fig. 15. Jacobi and CGS convergence behaviour for the Courier model with $k = 4$

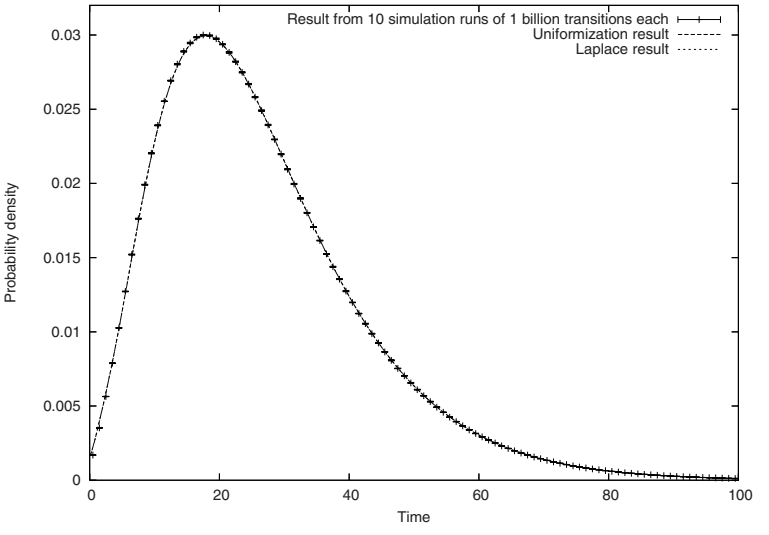


Fig. 16. Numerical and simulated response time densities for time taken from the initiation of a transport layer transmission (i.e. those markings for which $M(p_{11}) > 0$) to the arrival of an acknowledgement packet (i.e. those markings for which $M(p_{20}) > 0$). The median response time (50% quantile) is 0.0048 seconds, and the 95% quantile is 0.0114 seconds.

are also given. Once again the numerical results are compared against a simulation, and agreement is excellent.

For this example, our Laguerre scaling algorithm selected a damping parameter of $\sigma = 0.008$. A single slave (a 1.4GHz Athlon processor with 256MB RAM) required 24 minutes 15 seconds to calculate the 200 points plotted on the numerical passage time density graph. Using 8 slave PCs with the same configuration decreased the required time to just 3 minutes 23 seconds (corresponding to an efficiency of 96%). 16 slave PCs required 2 minutes 17 seconds (72% efficiency). These results reflect the excellent scalability of our approach.

5.2 Voting Model

Description. Fig. 17 represents a voting system with CC voters, MM polling units and NN central voting servers. In this system, voters cast votes through polling units which in turn register votes with all available central voting units. Both polling units and central voting units can suffer breakdowns, from which there is a soft recovery mechanism. If, however, all the polling or voting units fail, then, with high priority, a failure recovery mode is instituted to restore the system to an operational state.

We demonstrate the SMP passage-time analysis techniques of the previous sections with a large semi-Markov model of a distributed voting system (Fig. 17). The model is specified in a semi-Markov stochastic Petri net (SM-SPN)

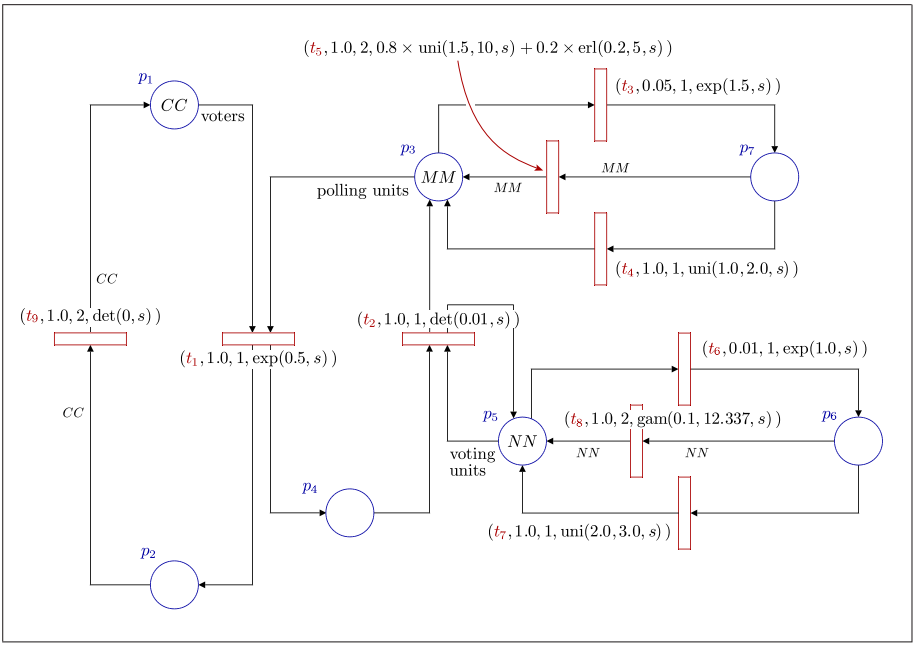


Fig. 17. A semi-Markov stochastic Petri net of a voting system with breakdowns and repairs

formalism [67] using an extension of the DNAmaca Markov chain modelling language [62].

The distributions are specified directly as Laplace transforms with certain macros provided for popular distributions (e.g. uniform, gamma, deterministic) and can be made marking dependent by use of the $m(p_i)$ function (which returns the current number of tokens at place, p_i). Support for inhibiting transitions is also provided.

For the voting system, Tab. 5 shows how the size of the underlying SMP varies according to the configuration of the variables CC, MM, and NN.

First Passage Time Analysis. The results presented in this section were produced on a Beowulf Linux cluster with 64 dual processor nodes, a maximum of 34 of which can be used by a single job. Each node has two Intel Xeon 2.0GHz processors and 2GB of RAM. The nodes are connected by a Myrinet network with a peak throughput of 250 Mb/s.

We display passage time densities produced by the iterative passage time algorithm and also by simulation to validate those results.

Fig. 18 shows the density of the time taken to process 300 voters (as given by the passage of 300 tokens from place p_1 to p_2) in system 6 of the voting model. Calculation of the analytical density required 15 hours and 7 minutes using 64 slave processors (in 8 groups of 8) for the 31 t-points plotted. Our

Table 5. Different configurations of the voting system and state space generated

System	CC	MM	NN	States
1	60	25	4	106 540
2	100	30	4	249 760
3	125	40	4	541 280
4	150	40	5	778 850
5	175	45	5	1 140 050
6	300	80	10	10 999 140

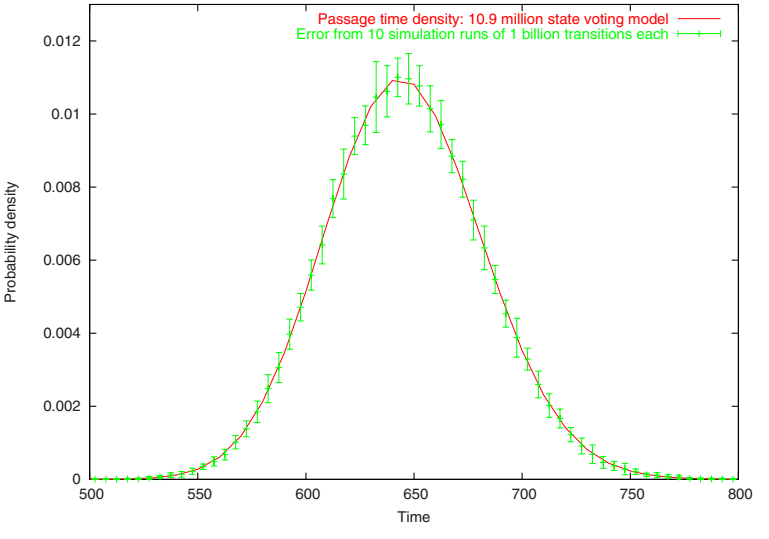


Fig. 18. Analytic and simulated (with 95% confidence intervals) density for the time taken to process 300 voters in the voting model system 6 (10.9 million states)

algorithm evaluated $L_{ij}(s)$ at 1 023 s-points, each of which involved manipulating sparse matrices of rank 10 999 140. The analytical curve is validated against the combined results from 10 simulations, each of which consisted of 1 billion transition firings. Despite this large simulation effort, we still observe wide confidence intervals (probably because of the rarity of source states).

Fig. 19 is a cumulative distribution for the same passage as Fig. 18 (easily obtained by inverting $L_{ij}(s)/s$ from cached values of $L_{ij}(s)$). It allows us to extract response time quantiles, for instance:

$$\mathbb{P}(\text{system 6 can process 300 voters in less than 730 seconds}) = 0.9876$$

5.3 PEPA Active Badge Model

Description. In the original active badge model, described in [68], there are 4 rooms on a corridor, all installed with active badge sensors, and a single person

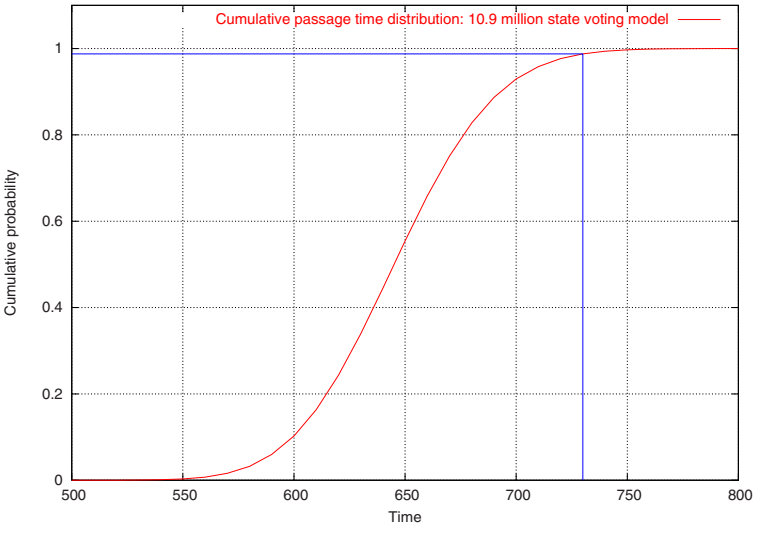


Fig. 19. Cumulative distribution function and quantile of the time taken to process 300 voters in the voting model system 6 (10.9 million states)

who can move from one room to an adjacent room. The sensors are linked to a database which records which sensor has been activated last. In the model of Fig. 20, we have M people in N rooms with sensors and a database that can be in one of N states. To maintain a reasonable state space, this is a simple database which does not attempt to keep track of every individual's location; rather it remembers the last movement that was made by any person in the system.

In the model below, $Person_i$ represents a person in room i, $Sensor_i$ is the sensor in room i and $Dbase_i$ is the state of the database. A person in room i can either move to room $i-1$ or $i+1$ or, if they remain there long enough, set off the sensor in room i, which registers its activation with the database.

The first thing to note about such a model is how fast the state space can grow. With M people in N rooms, we already have N^M states just from the different configurations of people in rooms. Then there are 2^N sensor configurations and finally N states that the database can be in, giving us a total of $2^N N^{M+1}$ states. For as few as 3 people and 6 rooms, the example we use, we have a global state space of $82,944$ states.

First Passage Time Analysis. We include two passages from the active badge system with 3 people and 6 possible rooms. As the model of Fig. 20 tells us, all 6 people start in room 1 and move out from there.

Fig. 21 shows the density function for the passage representing how long it takes for all 3 people to be together in room 6 for the first time.

It is interesting to observe that it is virtually impossible for all 3 people to end up in room 6, which requires 6 successive *move* transitions from all 3 people for it to happen at the earliest opportunity, until at least 10 time units have elapsed.

$$
\begin{aligned}
Person_1 &= (reg_1, r).Person_1 + (move_2, m).Person_2 \\
Person_i &= (move_{i-1}, m).Person_{i-1} + (reg_i, r).Person_i \\
&\quad + (move_{i+1}, m).Person_{i+1} \\
&\quad : 1 < i < N \\
Person_N &= (move_{N-1}, m).Person_{N-1} + (reg_N, r).Person_N \\
\\
Sensor_i &= (reg_i, \top).(rep_i, s).Sensor_i \quad : 1 \leq i \leq N \\
\\
Dbase_i &= \sum_{j=1}^{N} (rep_j, \top).Dbase_j \quad : 1 \leq i \leq N \\
\\
Sys &= \prod_{j=1}^{M} Person_1 \underset{Reg}{\bowtie} \prod_{j=1}^{N} Sensor_j \underset{Rep}{\bowtie} Dbase_1
\end{aligned}
$$

where $Reg = \{reg_i \mid 1 \leq i \leq N\}$ and $Rep = \{rep_i \mid 1 \leq i \leq N\}$

Fig. 20. The PEPA description for the generalised active badge model with N rooms and M people

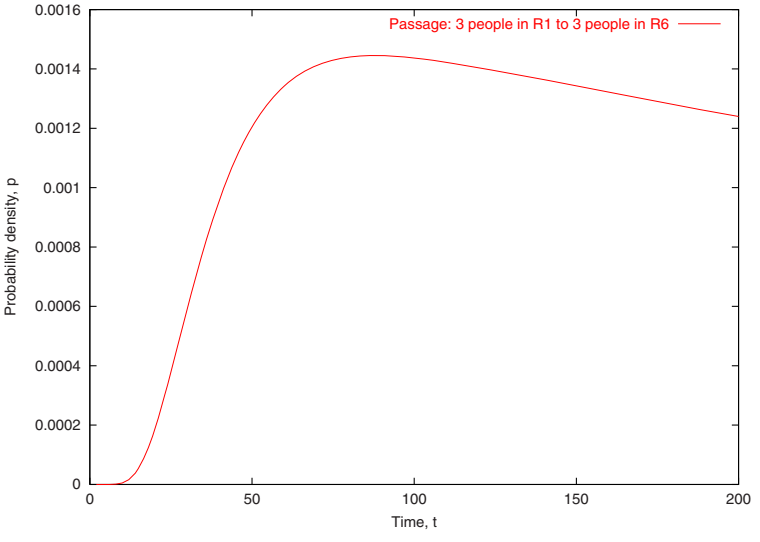


Fig. 21. The passage time density for 3 people starting in room 1 ending up all in room 6

After that time, very low probabilities are registered and the distribution clearly has a very heavy tail.

The second passage of Fig. 22 shows an equivalent passage time density from the same start point to a terminating condition of at least one person of the three entering room 6. The resulting passage is much less heavy tailed, as this time only a single person has to make it to room 6 before the passage ends.

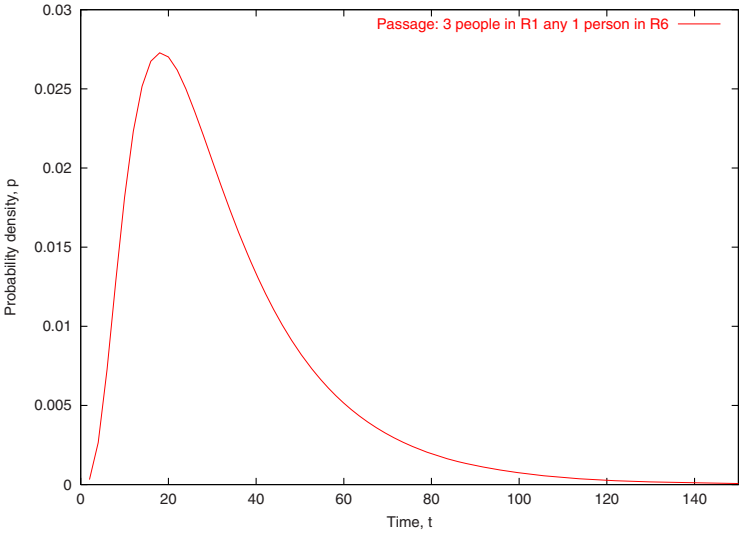


Fig. 22. The passage time density for 3 people starting in room 1 ending up with any one or more of them in room 6

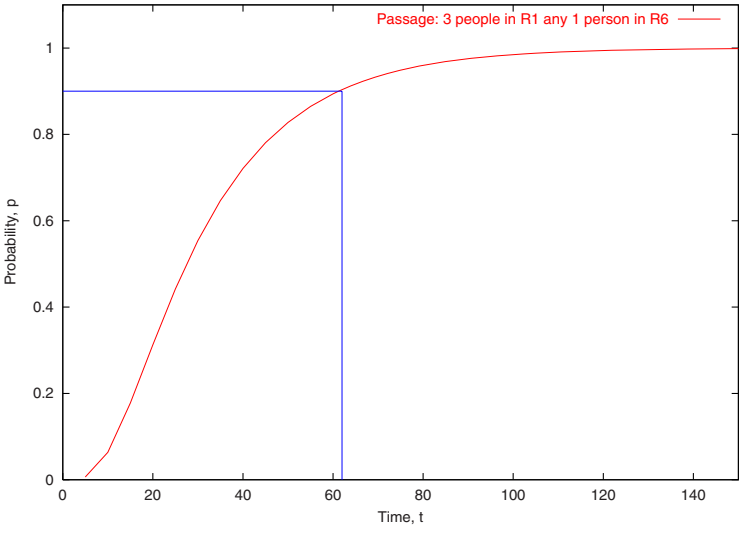


Fig. 23. The cumulative passage time distribution function for 3 people starting in room 1 ending up with any one or more of them in room 6

From these densities, it is a simple matter to construct cumulative distribution functions (the integral of the density function) and obtain quantiles, e.g. the probability that 3 people all reach room 6 by time $t = 150$. Fig. 23 shows the cumulative distribution function (cdf) corresponding to the passage time density

of Fig. 22. From this cdf, we can ascertain, for example, that there is a 90% probability that at least one person will have reached room 6 by time $t = 62$.

6 Conclusion and Future Perspectives

Performance analysis of complex systems is a computationally expensive activity. If a model does not have exploitable symmetries or other structure that allows for analytical or numerical shortcuts to be used, then an explicit representation of the state space has to be constructed. This chapter has discussed some state space generation methods and numerical algorithms for steady-state and passage-time analysis of (semi-)Markov models which are scalable across large computing clusters. We have shown that by making use of probabilistic algorithms and efficient distribution strategies (e.g. using hypergraph partitioning), we can subdivide large performance analysis problems in such a way that makes them tractable on individual computer nodes.

An important emerging development with the potential to tackle exceptionally large state spaces is the use of continuous approximations to represent large discrete state spaces. Preliminary efforts to relate this to modelling formalisms have led to continuous state-space translations from SPNs [69] and PEPA [70]. In both cases, repeated structures in the top-level formalism are represented by systems of ordinary differential equations (ODEs) which describe a deterministic trace of behaviour. In certain structural situations [71] the steady-state solution of the ODEs corresponds to the steady-state solution of the underlying Markov chain.

Acknowledgements

The authors would like to thank Nicholas Dingle, Peter Harrison and Aleksandar Trifunovic who contributed hugely to the development of the work presented here and without whom this would not have been possible.

References

1. J. Yang, C. Sar, and D. Engler, "eXplode: a Lightweight, General System for Finding Serious Storage System Errors," in *Proc. 7th Symposium on Operating System Design and Implementation*, (Seattle, WA), pp. 131–146, November 2006.
2. F. Bause and P. Kritzinger, *Stochastic Petri Nets – An Introduction to the Theory.* Wiesbaden, Germany: Verlag Vieweg, 1995.
3. W. Grassman, "Means and variances of time averages in Markovian environments," *European Journal of Operational Research*, vol. 31, no. 1, pp. 132–139, 1987.
4. A. Reibman and K. Trivedi, "Numerical transient analysis of Markov models," *Computers and Operations Research*, vol. 15, no. 1, pp. 19–36, 1988.
5. G. Bolch, S. Greiner, H. Meer, and K. Trivedi, *Queueing Networks and Markov Chains.* Wiley, August 1998.

6. B. Melamed and M. Yadin, "Randomization procedures in the computation of cumulative-time distributions over discrete state Markov processes," *Operations Research*, vol. 32, pp. 926–944, July–August 1984.
7. A. Miner, "Computing response time distributions using stochastic Petri nets and matrix diagrams," in *Proceedings of the 10th International Workshop on Petri Nets and Performance Models (PNPM'03)*, (Urbana-Champaign, IL), pp. 10–19, September 2nd–5th 2003.
8. J. Muppala and K. Trivedi, "Numerical transient analysis of finite Markovian queueing systems," in *Queueing and Related Models* (U. Bhat and I. Basawa, eds.), pp. 262–284, Oxford University Press, 1992.
9. M. Ajmone-Marsan, G. Conte, and G. Balbo, "A class of Generalised Stochastic Petri Nets for the performance evaluation of multiprocessor systems," *ACM Transactions on Computer Systems*, vol. 2, pp. 93–122, 1984.
10. W. Knottenbelt, *Parallel Performance Analysis of Large Markov Models*. PhD thesis, Imperial College London, February 2000.
11. J. T. Bradley, N. J. Dingle, P. G. Harrison, and W. J. Knottenbelt, "Distributed computation of passage time quantiles and transient state distributions in large semi-Markov models," in *PMEO'03, Performance Modelling, Evaluation and Optimization of Parallel and Distributed Systems*, (Nice), p. 281, IEEE Computer Society Press, April 2003.
12. C. Hoare, *Communicating Sequential Processes*. Prentice Hall, 1985.
13. R. Milner, *Communication and Concurrency*. Prentice Hall, 1989.
14. J. Hillston, *A Compositional Approach to Performance Modelling*. PhD thesis, University of Edinburgh, 1994.
15. M. Rettelbach and M. Siegle, "Compositional minimal semantics for the stochastic process algebra TIPP," in *Proceedings of the 2nd Workshop on Process Algebras and Performance Modelling*, pp. 31–50, Regensberg/Erlangen: Arbeitsberichte des IMMD, Universität Erlangen-Nürnberg, July 1994.
16. H. Hermanns and M. Rettelbach, "Syntax, semantics, equivalences and axioms for MTIPP," in *Proceedings of the 2nd Workshop on Process Algebras and Performance Modelling*, Regensberg/Erlangen: Arbeitsberichte des IMMD, Universität Erlangen-Nürnberg, July 1994.
17. P. Buchholz, "Markovian Process Algebra: composition and equivalence," in *Proceedings of the 2nd Workshop on Process Algebras and Performance Modelling*, Regensberg/Erlangen: Arbeitsberichte des IMMD, Universität Erlangen-Nürnberg, July 1994.
18. M. Bernardo, L. Donatiello, and R. Gorrieri, "Modelling and analyzing concurrent systems with MPA," in *Proceedings of the 2nd Workshop on Process Algebras and Performance Modelling*, pp. 89–106, Regensberg/Erlangen: Arbeitsberichte des IMMD, Universität Erlangen-Nürnberg, July 1994.
19. H. Hermanns, U. Herzog, and J. Hillston, "Stochastic process algebras—A formal approach to performance modelling," tutorial, Department of Computer Science, University of Edinburgh, Edinburgh EH9 3JZ, UK, 1996.
20. B. Strulo, *Process Algebra for Discrete Event Simulation*. PhD thesis, Imperial College, London, October 1993.
21. P. G. Harrison and B. Strulo, "SPADES - a process algebra for discrete event simulation," *Journal of Logic and Computation*, vol. 10, pp. 3–42, January 2000.
22. J. T. Bradley, "Semi-Markov PEPA: Modelling with generally distributed actions," *International Journal of Simulation*, vol. 6, pp. 43–51, January 2005.

23. M. Bravetti, M. Bernardo, and R. Gorrieri, "Towards performance evaluation with general distributions in process algebras," in *CONCUR'98, Proceedings of the 9th International Conference on Concurrency Theory* (D. Sangiorgi and R. de Simone, eds.), vol. 1466 of *Lecture Notes in Computer Science*, pp. 405–422, Springer-Verlag, Nice, September 1998.
24. M. Bravetti and R. Gorrieri, "Interactive generalized semi-Markov processes," in *Process Algebra and Performance Modelling Workshop* (J. Hillston and M. Silva, eds.), pp. 83–98, Centro Politécnico Superior de la Universidad de Zaragoza, Prensas Universitarias de Zaragoza, Zaragoza, September 1999.
25. J. Hillston and M. Ribaudo, "Stochastic process algebras: a new approach to performance modelling," in *Modelling and Simulation of Advanced Computer Systems* (K. Bagchi and G. Zobrist, eds.), ch. 10, pp. 235–256, Gordon Breach, 1998.
26. R. Bryant, "Graph-based algorithms for Boolean function manipulation," *IEEE Transactions on Computers*, vol. C-35, pp. 677–691, August 1986.
27. G. Ciardo and A. Miner, "A data structure for the efficient Kronecker solution of GSPNs," in *Proceedings of the 8th International Conference on Petri Nets and Performance Models (PNPM'99)*, (Zaragoza, Spain), pp. 22–31, IEEE Computer Society Press, September 1999.
28. A. Miner and D. Parker, "Symbolic representations and analysis of large probabilistic systems," in *Lecture Notes in Computer Science 2925/2004: Validation of Stochastic Systems*, pp. 296–338, 2004.
29. P. Buchholz and P. Kemper, "Kronceker based matrix representations for large Markov models," in *Lecture Notes in Computer Science 2925/2004: Validation of Stochastic Systems*, pp. 256–295, 2004.
30. M. Rabin, "Probabilistic algorithm for testing primality," *Journal of Number Theory*, vol. 12, pp. 128–138, 1980.
31. M. Kuntz and K. Lampka, "Probabilistic meothds in state space analysis," in *Lecture Notes in Computer Science 2925/2004: Validation of Stochastic Systems*, pp. 339–383, 2004.
32. G. Holzmann, *Design and Validation of Computer Protocols*. Prentice-Hall, 1991.
33. G. Holzmann, "An analysis of bitstate hashing," in *Proceedings of IFIP/PSTV95: Conference on Protocol Specification, Testing and Verification*, Warsaw, Poland: Chapman & Hall, June 1995.
34. B. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of the ACM*, vol. 13, pp. 422–426, July 1970.
35. P. Wolper and D. Leroy, "Reliable hashing without collision detection," in *Lecture Notes in Computer Science 697*, pp. 59–70, Springer Verlag, 1993.
36. U. Stern and D. Dill, "Improved probabilistic verification by hash compaction," in *IFIP WG 10.5 Advanced Research Working Conference on Correct Hardware Design and Verification Methods*, 1995.
37. W. J. Knottenbelt, P. G. Harrison, M. S. Mestern, and P. S. Kritzinger, "A probabilistic dynamic technique for the distributed generation of very large state spaces," *Performance Evaluation*, vol. 39, pp. 127–148, February 2000.
38. M. Raynal, *Distributed Algorithms and Protocols*. John Wiley and Sons, 1988.
39. E. Dijkstra, W. Feijen, and A. Gasteren, "Derivation of a termination detection algorithm for distributed computations.," *Information Processing letters*, vol. 16, pp. 217–219, June 1983.
40. W. Stewart, *Introduction to the Numerical Solution of Markov Chains*. Princeton University Press, 1994.

41. R. Weiss, "A theoretical overview of Krylov subspace methods," *Applied Numerical Mathematics*, vol. 19, pp. 207–233, 1995. Special Issue on Iterative Methods for Linear Equations.
42. P. Sonneveld, "CGS, a fast Lanczos-type solver for nonsymmetric linear systems," *SIAM Journal on Scientific and Statistical Computing*, vol. 10, pp. 36–52, January 1989.
43. H. Vorst, "Bi-CGSTAB: A fast and smoothly converging variant of BiCG for the solution of nonsymmetric linear systems," *SIAM Journal on Scientific and Statistical Computing*, vol. 13, pp. 631–644, March 1992.
44. R. Freund, "A transpose-free quasi-minimal residual algorithm for non-Hermitian linear systems," *SIAM Journal on Scientific Computing*, vol. 14, pp. 470–482, March 1993.
45. D. D. Deavours and W. H. Sanders, "An efficient disk-based tool for solving very large Markov models," in *TOOLS 1997, Computer Performance Evaluation: Modelling Techniques and Tools*, vol. 1245 of *Lecture Notes in Computer Science*, (St. Malo), pp. 58–71, Springer-Verlag, June 1997.
46. D. Deavours and W. Sanders, "An efficient disk-based tool for solving large Markov models," *Performance Evaluation*, vol. 33, pp. 67–84, June 1998.
47. M. Kwiatkowska and R. Mehmood, "Out-of-core solution of large linear systems of equations arising from stochastic modelling," in *Proceedings of Process Algebra and Performance Modelling (PAPM'02)*, (Copenhagen), pp. 135–151, July 25th–26th 2002.
48. R. Mehmood, "Serial disk-based analysis of large stochastic models," in *Lecture Notes in Computer Science 2925/2004: Validation of Stochastic Systems*, pp. 230–255, 2004.
49. W. J. Knottenbelt and P. G. Harrison, "Distributed disk-based solution techniques for large Markov models," in *NSMC'99, Proceedings of the 3rd Intl. Conference on the Numerical Solution of Markov Chains*, (Zaragoza), pp. 58–75, September 1999.
50. A. Bell and B. Haverkort, "Serial and parallel out-of-core solution of linear systems arising from Generalised Stochastic Petri Nets," in *Proc. High Performance Computing Symposium (HPC 2001)*, pp. 242–247, 2001.
51. U. Catalyürek and C. Aykanat, "Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication," *IEEE Transactions on Parallel and Distributed Systems*, vol. 10, pp. 673–693, July 1999.
52. G. Karypis and V. Kumar, *hMETIS: A Hypergraph Partitioning Package, Version 1.5.3*. University of Minnesota, November 1998.
53. A. Trifunovic and W. Knottenbelt, "Parkway 2.0: A parallel multilevel hypergraph partitioning tool," in *Proceedings of the 19th International Symposium on Computer and Information Sciences (ISCIS'04)*, (Antalya, Turkey), October 27th–29th 2004.
54. P. G. Harrison, "Laplace transform inversion and passage-time distributions in Markov processes," *Journal of Applied Probability*, vol. 27, pp. 74–87, March 1990.
55. J. Abate and W. Whitt, "The Fourier-series method for inverting transforms of probability distributions," *Queueing Systems*, vol. 10, no. 1, pp. 5–88, 1992.
56. P. G. Harrison and W. Knottenbelt, "Passage time distributions in large Markov chains," in *Proceedings of ACM SIGMETRICS 2002*, (Marina Del Rey, California), pp. 77–85, June 2002.
57. J. Abate and W. Whitt, "Numerical inversion of Laplace transforms of probability distributions," *ORSA Journal on Computing*, vol. 7, no. 1, pp. 36–43, 1995.

58. J. Abate, G. L. Choudhury, and W. Whitt, "On the Laguerre method for numerically inverting Laplace transforms," *INFORMS Journal on Computing*, vol. 8, no. 4, pp. 413–427, 1996.
59. P. G. Harrison and W. J. Knottenbelt, "Passage-time distributions in large Markov chains," in *Proceedings of ACM SIGMETRICS 2002* (M. Martonosi and E. A. de Souza e Silva, eds.), pp. 77–85, Marina Del Rey, USA, June 2002.
60. J. T. Bradley, N. J. Dingle, W. J. Knottenbelt, and H. J. Wilson, "Hypergraph-based parallel computation of passage time densities in large semi-Markov models," *Journal of Linear Algebra and Applications*, vol. 386, pp. 311–334, July 2004.
61. N. Dingle, *Parallel Computation of Response Time Densities and Quantiles in Large Markov and Semi-Markov Models*. PhD thesis, Imperial College London, October 2004.
62. W. Knottenbelt, "Generalised Markovian analysis of timed transition systems," Master's thesis, University of Cape Town, Cape Town, South Africa, July 1996.
63. W. Gropp, E. Lusk, and A. Skjellum, *Using MPI: Portable Parallel Programming with the Message Passing Interface*. Cambridge, Massachussetts: MIT Press, 1994.
64. C. Woodside and Y. Li, "Performance Petri net analysis of communication protocol software by delay-equivalent aggregation," in *Proceedings of the 4th International Workshop on Petri nets and Performance Models (PNPM'91)*, (Melbourne, Australia), pp. 64–73, IEEE Computer Society Press, 2–5 December 1991.
65. W. Gropp, E. Lusk, and A. Skjellum, *Using MPI: Portable Parallel Programming with the Message Passing Interface*. Cambridge, Massachussetts: MIT Press, 1994.
66. A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam, *PVM Parallel Virtual Machine: A Users' Guide and Tutorial for Networked Parallel Computing*. Cambridge, Massachussetts: MIT Press, 1994.
67. J. T. Bradley, N. J. Dingle, W. J. Knottenbelt, and P. G. Harrison, "Performance queries on semi-Markov stochastic Petri nets with an extended Continuous Stochastic Logic," in *PNPM'03, Proceedings of Petri Nets and Performance Models* (G. Ciardo and W. Sanders, eds.), (University of Illinois at Urbana-Champaign), pp. 62–71, IEEE Computer Society, September 2003.
68. S. Gilmore, J. Hillston, and G. Clark, "Specifying performance measures for PEPA," in *Proceedings of the 5th International AMAST Workshop on Real-Time and Probabilistic Systems*, vol. 1601 of *Lecture Notes in Computer Science*, (Bamberg), pp. 211–227, Springer-Verlag, 1999.
69. J. Julvez, E. Jimenez, L. Recalde, and M. Silva, "On observability in timed continuous Petri net systems," in *QEST'04, Proceedings of 1st International Conference on the Quantitative Evaluation of Systems*, (Enschede), pp. 60–69, IEEE Computer Society Press, September 2004.
70. J. Hillston, "Fluid flow approximation of PEPA models," in *QEST'05, Proceedings of the 2nd International Conference on Quantitative Evaluation of Systems*, (Torino), pp. 33–42, IEEE Computer Society Press, September 2005.
71. T. G. Kurtz, "Solutions of ordinary differential equations as limits of pure jump Markov processes," *Journal of Applied Probability*, vol. 7, pp. 49–58, April 1970.

Data Representation and Efficient Solution: A Decision Diagram Approach*

Gianfranco Ciardo

Dept. of Computer Science and Engineering, UC Riverside, CA 92521, USA
ciardo@cs.ucr.edu

Abstract. Decision diagrams are a family of data structures that can compactly encode many functions on discrete structured domains, that is, domains that are the cross-product of finite sets. We present some important classes of decision diagrams and show how they can be effectively employed to derive "symbolic" algorithms for the analysis of large discrete-state models. In particular, we discuss both explicit and symbolic algorithms for state-space generation, CTL model-checking, and continuous-time Markov chain solution. We conclude with some suggestions for future research directions.

Keywords: binary decision diagrams, multi-valued decision diagrams, edge-valued decision diagrams, state-space generation, symbolic model checking, Kronecker algebra.

1 Introduction

Discrete-state models are useful to describe and analyze computer-based and communication systems, or distributed software, and many other man-made artifacts. Due to the inherent combinatorial nature of their interleaving behavior, such models can easily have enormous state spaces, which can make their computer-supported analysis very difficult. Of course, high-level formalisms such as Petri nets, queueing networks, communicating sequential processes, and specialized languages can be effectively used to describe these enormous underlying state space. However, when a model described in one of these formalisms needs to be analyzed, the size of the state space remains a major obstacle.

In this contribution, we present *decision diagrams*, a class of data structures that can compactly encode functions (or set and relations, or vectors and matrices) on very large but structured domains. Then, we briefly summarize some of the main computational tasks involved in the traditional "explicit" analysis of systems, both in a strictly logical setting and in a Markovian setting. Putting the two together, we then show how these tasks can be effectively performed "symbolically" using decision diagrams. We conclude by listing some research challenges lying ahead.

* Work supported in part by the National Science Foundation under grants CNS-0501747 and ATM-0428880.

M. Bernardo and J. Hillston (Eds.): SFM 2007, LNCS 4486, pp. 371–394, 2007.

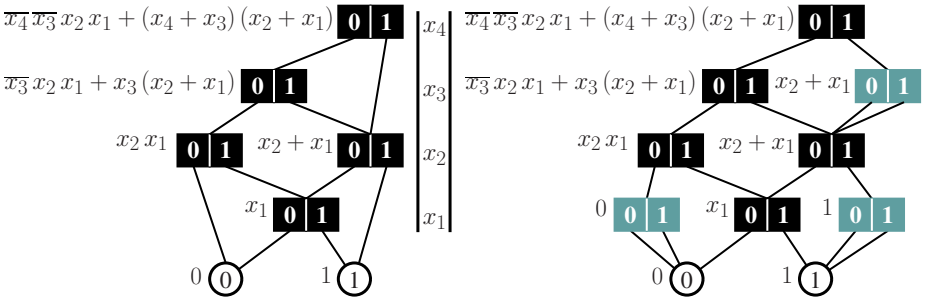


Fig. 1. Fully-reduced vs. quasi-reduced BDDs

In the following, calligraphic letters (e.g., $\mathcal{A}$, $\mathcal{X}$) denote sets or relations, lowercase bold letters (e.g., $\mathbf{i}$, $\mathbf{j}$) indicate global state of the system, and lowercase italic letters (e.g., i, x_k) indicate local components of the state. Also, $\mathbb{B}$, $\mathbb{N}$, $\mathbb{Z}$, and $\mathbb{R}$ indicate the set $\{0, 1\}$ of boolean values, the natural numbers, the integers, and the real numbers, respectively.

2 Decision Diagrams: A Data Structure for Structured Data

Many classes of decision diagrams have been defined in the literature. This section presents some of the most common ones, which will be used in Section 4 to improve the efficiency of discrete-state system analysis.

2.1 Binary Decision Diagrams

The most widely known and used class of decision diagrams is by far the **binary decision diagrams (BDDs)** [3], which provide a compact representation for functions of the form $f : \mathbb{B}^L \to \mathbb{B}$, for some finite $L \in \mathbb{N}$. In particular, if the BDDs are *reduced* and *ordered* [3,4], properties we assume from now on for all classes of decision diagrams we discuss, they enjoy several important advantages. Such BDDs are *canonical*, thus testing for satisfiability, i.e.,

"is there an $\mathbf{i} \in \mathbb{B}^L$ such that $f(\mathbf{i}) = 1$?"

or equivalence, i.e.,

"given functions f and g, is $f(\mathbf{i}) = g(\mathbf{i})$ for all $\mathbf{i} \in \mathbb{B}^L$?"

can be done in constant time, while important binary operations such as conjunction, disjunction, and relational product (described in detail later) require time and memory proportional to the product of the size (number of nodes) of the argument BDDs, in the worst case.

Formally, an L-variable BDD is an acyclic directed edge-labeled graph where each of its nodes encodes a function of the form $f : \mathbb{B}^L \to \mathbb{B}$. The nodes of the graph are assigned to a level, we let $p.lvl$ denote the level of node p. A non-terminal node at level k, with $L \geq k \geq 1$, corresponds to a choice for the value of the boolean variable x_k, the k^{th} argument to the function, while the two terminal nodes 0 and 1 correspond to the two possible values of the function. A node p at level k has two edges, labeled with the possible values of x_k, 0 and 1. The edge labeled 0, or 0-edge, points to node $p[0]$ while the 1-edge point to node $p[1]$, where both nodes are at levels below k (this is the "ordered" property: nodes are found along any path in an order consistent with the order $x_L, ..., x_1$ of the argument variables). Also, nodes must be *unique* (thus, no node can be a *duplicate* of another node at the same level, i.e., have have the same 0-child and the same 1-child) and *non-redundant*, i.e., $p[0]$ and $p[1]$ must differ (this is the "fully reduced" property). A node p at level k encodes the function $v_p : \mathbb{B}^L \to \mathbb{B}$ defined recursively by

$$v_p(x_L, ..., x_1) = \begin{cases} p & \text{if } k = 0 \\ \overline{x_k} \wedge v_{p[0]}(x_L, ..., x_1) \ \vee \ x_k \wedge v_{p[1]}(x_L, ..., x_1) & \text{if } k > 0. \end{cases}$$

Thus, given a constant vector $\mathbf{i} = (i_L, ..., i_1) \in \mathbb{B}^L$, we can evaluate $v_p(\mathbf{i})$ in $O(L)$ time. Fig. 1 on the left shows an example of BDD and the boolean functions encoded by its nodes. On the right, the same set of functions are encoded using a "quasi-reduced" version of BDDs , where duplicate nodes are not allowed, but redundant nodes (shown in gray in the figure) may have to be present, since edges can only span one level. Such BDDs are still canonical, and, while they may use more nodes, all their edges connect nodes at adjacent levels, resulting in simpler manipulation algorithms.

Strictly speaking, a BDD encoding a given function f has a specific root p such that $f = v_p$. In practice, BDD algorithms need to manage multiple functions on the same domain $\mathbb{B}^L$, and this is done by storing (without duplicates) all the roots of these functions, as well as the nodes reached by them, in a single BDD, often referred to as a BDD *forest*.

2.2 Multi-valued, Multi-terminal, and Multi-dimensional Extensions

Many variants of BDDs have been defined to extend their applicability or to target specific applications. This section discusses several "terminal-valued" variants, that is, decision diagrams where the value of function f evaluated on argument $\mathbf{i}$ is given by the the terminal node reached when following the path corresponding to $\mathbf{i}$, just as is the case for BDDs.

Multi-valued decision diagrams (MDDs) [24] encode functions of the form $\widehat{\mathcal{X}} \to \mathbb{B}$, where the domain $\widehat{\mathcal{X}}$ is the cross-product $\widehat{\mathcal{X}} = \mathcal{X}_L \times \cdots \times \mathcal{X}_1$ of L finite sets and each $\mathcal{X}_k$, for $L \geq k \geq 1$, is of the form $\mathcal{X}_k = \{0, 1, ..., n_k - 1\}$, for some $n_k \in \mathbb{N}$. Thus, a non-terminal node at level k corresponds to a multi-way

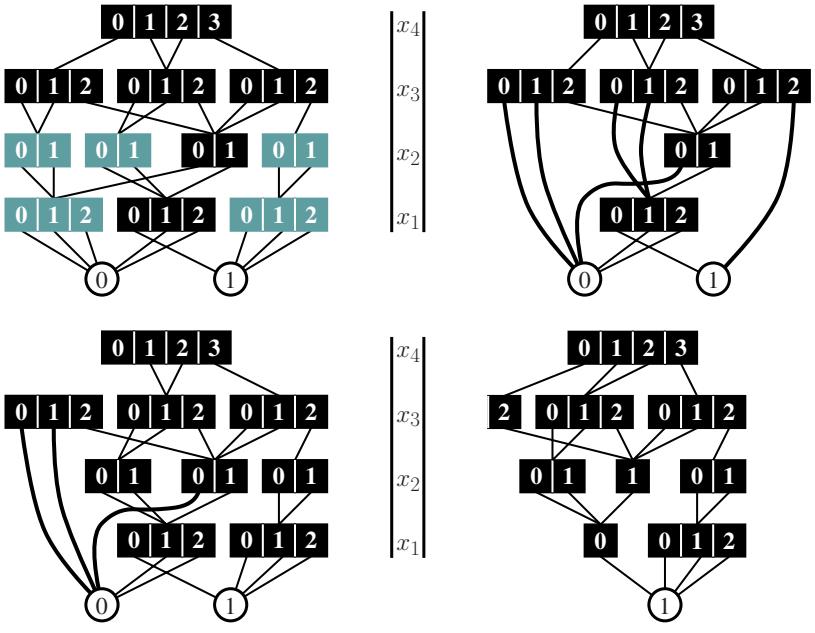


Fig. 2. Fully-reduced vs. quasi-reduced, full vs. sparse representation of MDDs

choice for the argument variable x_k. The top of Fig. 2 shows, on the left, a quasi-reduced MDD where redundant nodes p with $p[0] = \cdots = p[n_k - 1]$ are kept and, on the right, the same MDD in a fully-reduced version, where its "long edges" (spanning multiple levels) are shown with thicker lines. The bottom of the same figure shows, on the left a *default-reduced* version of the same MDD where only edges to the default terminal node 0 can span multiple levels (a still canonical compromise between the fully-reduced and quasi-reduced versions) and, on the right, its *sparse* representation where paths leading to node 0 are not shown. Graphically, this last representation is quite compact, and it also reflects quite closely how MDDs are implemented in the tool SMART [9].

Multi-terminal BDDs (MTBDDs) [21] can encode functions of the form $\mathbb{B}^L \to \mathbb{R}$, by attaching arbitrary values from $\mathbb{R}$ to the terminal nodes of a binary decision diagram. The **algebraic decision diagrams (ADDs)** [1] are exactly analogous, except were defined to encode function on arbitrary ranges, not just the reals. The **multi-terminal multi-valued decision diagrams (MT-MDDs)**, an example of which is shown in Fig. 3 on the left in fully-reduced version and on the right in quasi-reduced version, naturally extend MTBDDs by additionally allowing multi-way choices at each node.

A function $f : \widehat{\mathcal{X}} \to \mathcal{S}$ can of course also be thought of as an $\mathcal{S}$-valued one-dimensional vector of size $|\widehat{\mathcal{X}}|$. Many applications also need to encode functions of the form $\widehat{\mathcal{X}} \times \widehat{\mathcal{X}} \to \mathcal{S}$, or two-dimensional matrices. An obvious way to

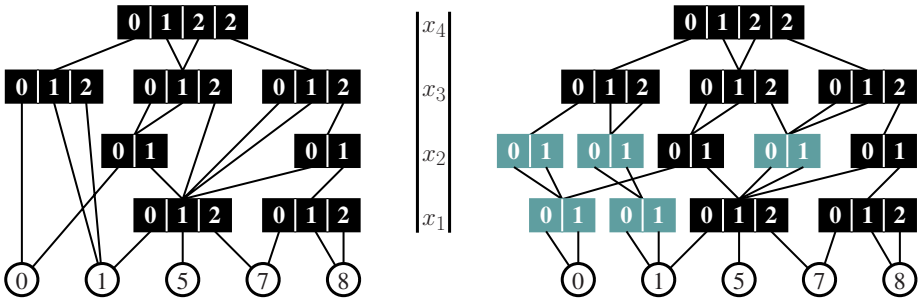


Fig. 3. Fully-reduced vs. quasi-reduced MTMDD

accomplish this is to use a BDD, MDD, MTBDD, or MTMDD with twice as many levels. The traditional notation uses an "unprimed" x_k for the rows, or "from", variables, and a "primed" x'_k for columns, or "to" variables. Furthermore, the manipulation algorithms are more easily written, and usually much more efficient, if the levels are *interleaved*, that is, the order of the function parameters is $(x_L, x'_L, ..., x_1, x'_1)$. A similar, but more direct, way to encode such matrices is to use a **(boolean-valued) matrix diagrams (MxDs)** [13,27], where a non-terminal node P at level k, for $L \geq k \geq 1$, has $n_k \times n_k$ edges, so that $P[i_k, i'_k]$ points to the node to be reached when $x_k = i_k$ and $x'_k = i'_k$. The top left of Fig. 4 shows a $2L$-level MDD, where $L = 2$, in sparse format. The encoded $(3 \cdot 2) \times (3 \cdot 2)$ matrix has zero entries in all but seven positions (the rows and columns of the resulting matrix are indexed by $x_2 \cdot 2 + x_1$ and $x'_2 \cdot 2 + x'_1$, respectively):

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	0	0	1	0
2	0	0	0	0	0	0
3	1	1	1	1	0	0
4	0	0	1	0	0	0
5	0	0	0	1	0	0

where

$0 \equiv (x_2 = 0, x_1 = 0)$
$1 \equiv (x_2 = 0, x_1 = 1)$
$2 \equiv (x_2 = 1, x_1 = 0)$
$3 \equiv (x_2 = 1, x_1 = 1)$
$4 \equiv (x_2 = 2, x_1 = 0)$
$5 \equiv (x_2 = 2, x_1 = 1)$

MxDs, however, where not introduced just to stress the natural interleaving of unprimed and primed variables, but to exploit a much more fundamental property often present in large asynchronous systems: the large number of *identity patterns*. The top right of Fig. 4 show the MxD encoding the same matrix, and the gray node in it is an example of an identity: its diagonal edges points to the same node, the terminal node 1 in this case, while its off-diagonal entries point to node 0. The bottom left of Fig. 4 shows the *identity-reduced* version of MxDs which is commonly employed, where long edges signify skipped identity nodes; on the right is the sparse format representation, which just lists explicitly the row-column pairs of indices corresponding to non-zero node entries.

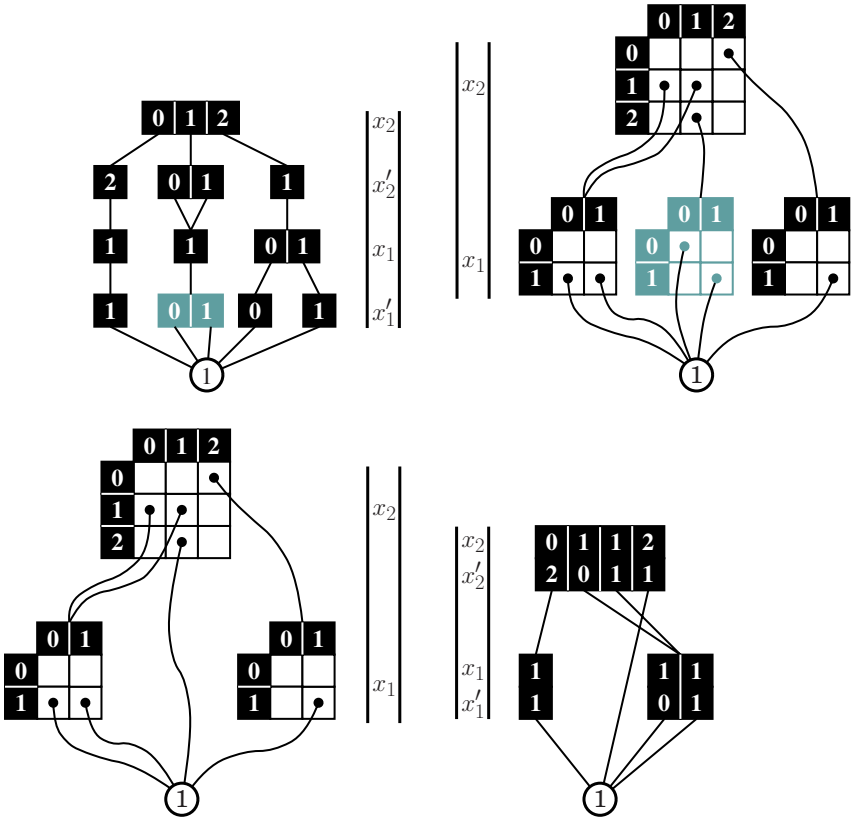


Fig. 4. Representing a boolean matrix with $2L$-level MDDs or with MxDs

2.3 Edge-Valued Extensions

Further "edge-valued" variants of decision diagrams have been defined to represent functions with a non-boolean range, as MTMDDs and ADDs can, but in such a way that the value of the function is not found in a terminal node, but is distributed over the edges found along a path. The manipulation algorithms are generally more complex, but these classes of decision diagrams can be exponentially more compact than their respective terminal-valued versions.

Edge-valued BDDs (EVBDDs) [26] encode functions of the form $\mathbb{B}^L \to \mathbb{Z}$, by having a single terminal node "Ω", which carries no value, and associating integer values to the edges of the diagram. The value of the function is obtained by *adding* the values encountered along the path to Ω corresponding the the function's argument; the result is a possibly exponentially smaller diagram than with an MTBDD. Nodes are normalized by scaling their edge values so that the 0-edge has an associated value of 0 (this fact can be used to save storage in a practical implementation, but is also one way to enforce canonicity), and the root

node has a "dangling arc" whose associated value is summed to the path value when evaluating the function, thus it is the value of the function being encoded when the argument is $(0, ..., 0)$. The **EVMDD** shown in Fig. 5 on the left is the corresponding version with multi-way choices, it encodes $f(x_4, x_3, x_2, x_1) = \sum_{L \geq k \geq 1} x_k \cdot \prod_{k > h \geq 1} n_h$, i.e., the value of (x_4, x_3, x_2, x_1) interpreted as a *mixed-base* integer; the same function would require a full tree, thus exponential space, if encoded with an MTMDD. Formally, the function $v_{(\sigma,p)} : \widehat{\mathcal{X}} \to \mathbb{Z}$ encoded by edge (σ, p), where $\sigma \in \mathbb{Z}$ and p is a node at level k, is defined recursively as

$$v_{(\sigma,p)}(x_L, ..., x_1) = \begin{cases} \sigma & \text{if } k = 0 \\ \sigma + v_{p[x_k]}(x_L, ..., x_1) & \text{if } k > 0. \end{cases}$$

The **positive edge-valued MDDs (EV$^+$MDDs)** [14] use a different normalization rule where all edge values leaving a node are non-negative or (positive) ∞, but at least one of them is zero, so that the value associated with the dangling edge is the minimum of the function. To ensure canonicity, an edge with an associated value of ∞ can only point to Ω. EV$^+$MDDs can encode arbitrary partial functions of the form $\widehat{\mathcal{X}} \to \mathbb{N} \cup \{\infty\}$. For example, the function encoded by the EV$^+$MDD in the middle of Fig. 5 cannot be encoded by an EVMDD because $f(0, ..., 0) = \infty$, but f is not identically ∞. Furthermore, if the dangling arc of the EV$^+$MDD is allowed to be an arbitrary integer, then , arbitrary partial functions of the form $\widehat{\mathcal{X}} \to \mathbb{Z} \cup \{\infty\}$ can be encoded. The EV$^+$MDD shown on the right of Fig. 5 shows the equivalent quasi-reduced version: the condition for a node to be redundant (as the additional gray nodes in the figure are) is now that all of its edges point to the same node and have the same associated value, which must then be 0, given the normalization requirement. In the figure, long edges with an associated value of ∞ can still exist; alternatively, even those could be required to span only one level through the introduction of further redundant node, but this would not further simplify the manipulation algorithms.

We already saw boolean MxDs, but, originally, **(real-valued) matrix diagrams (MxDs)** [13,27] were introduced to overcome some of the applicability and efficiency limitations encountered when using Kronecker algebra [18] to encode the transition rate matrix of large structured Markov chains [5,7,20]. These MxDs can encode functions of the form $\widehat{\mathcal{X}}^2 \to \mathbb{R}^{\geq 0}$, that is, non-negative matrices, by having a row and column choice at each node and multiplying (instead of summing, as for EVMDDs) the values encountered along a path to the single terminal node Ω. Canonicity is enforced by requiring that the minimum non-zero edge value in each node be 1, so that the value associated to the dangling arc is, again, the minimum of the function (alternatively, one can require that the maximum edge value be 1, so that the value associated to the dangling is the maximum of the function). Fig. 6 on the left shows a two-level MxD; when an edge value is zero, the edge goes to Ω and is not shown for clarity, while the semantic of an edge skipping a level is that a redundant node with an identity pattern is assumed. Thus, in the figure, $f(x_2, x'_2, x_1, x'_1)$ is 7 when $x_2 = 2$, $x'_2 = 1$, $x_1 = x'_1$, and it is 0 when $x_2 = 2$, $x'_2 = 1$, $x_1 \neq x'_1$. The center of the

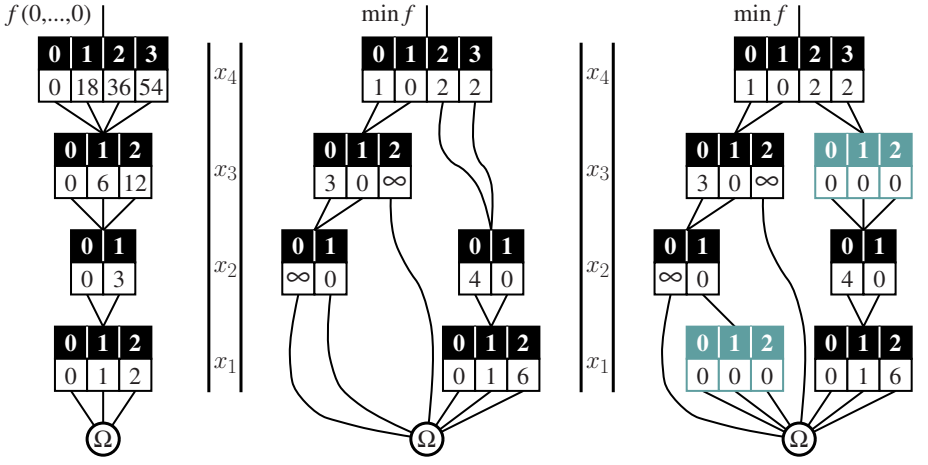


Fig. 5. EVMDDs, fully-reduced EV$^+$MDDs, and quasi-reduced EV$^+$MDDs

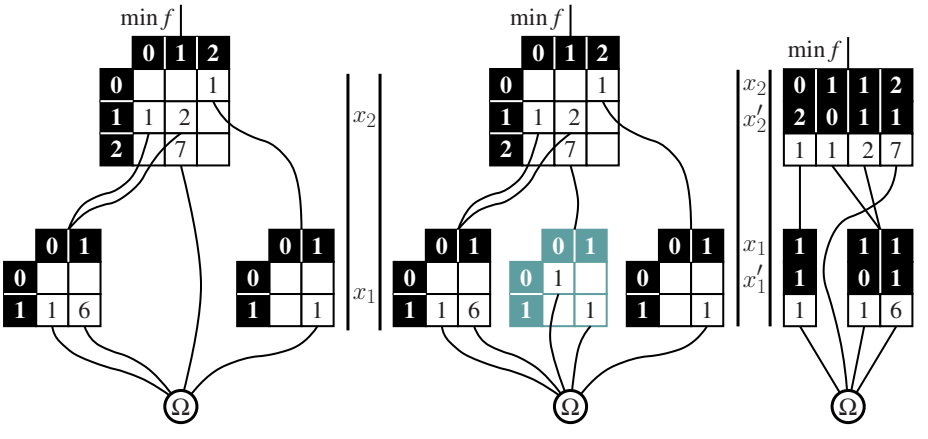


Fig. 6. Fully-reduced vs. quasi-reduced real-valued MxDs, and sparse representation

same figure shows the quasi-reduced version of this MxD, where the only long edges are those with an associated value of 0, thus are not shown, while the right shows its sparse representation.

2.4 Decision Diagram Manipulation Algorithms

So far we have discussed static aspects of various classes of decision diagrams, i.e., we have seen how decision diagrams can compactly encode functions over large structured domains. However, their time efficiency is just as important as their ability to save memory. Thus, we now turn to dynamic aspects of their manipulation. Decision diagram algorithms can usually be elegantly expressed in a recursive style. Two data structures are essential to achieve the desired

efficiency. First, a *unique table* is required to enforce canonicity. This is a hash table that can be searched based on the level and the pattern of edges (and values, in the edge-valued case) of a node, to avoid creating duplicate nodes. Second, an *operation cache* is used to look up whether a needed result of some operation on some nodes has been previously computed. Also this is a hash table, this time searched on the argument nodes' unique identifier (e.g., their memory address) and the operation code.

For example, Fig. 7 shows two BDD algorithms. The first one, *Or*, computes the disjunction of its arguments, i.e., given two functions $v_a, v_b : \mathbb{B}^L \rightarrow \mathbb{B}$, encoded by the two nodes a and b, it computes the node r encoding function v_r satisfying $v_r(\mathbf{i}) = v_a(\mathbf{i}) \vee v_b(\mathbf{i})$, for all $\mathbf{i} \in \mathbb{B}^L$. This algorithm assumes that these function are encoded in a fully-reduced BDD forest, thus, after testing for trivial cases that can be directly answered without recursion, it must test the level of a and b and proceed accordingly. Note that checking whether *Cache* contains already a result r for the *Or* of a and b is essential to efficiency; without it, the complexity would be proportional to the number of *paths* in the BDDs, not to the number of *nodes* in them.

The second algorithm in Fig. 7 computes the so-called *relational product* r of an L-level BDD x with a $2L$-level BDD t, i.e., $v_r(\mathbf{j}) = 1 \Leftrightarrow \exists \mathbf{i}, v_x(\mathbf{i}) = 1 \wedge v_t(\mathbf{i}, \mathbf{j}) = 1$. Note that, as the BDDs are assumed to be quasi-reduced, the recursion on x and t proceeds in lockstep, i.e., level k of x is processed with levels k and k' of t to compute level k (conceptually k') of the result, thus there is no need to check for the levels of the nodes, as in the fully-reduced case.

Edge-valued decision diagram algorithms also operate recursively, but the arguments passed and returned in the recursions are *edges*, i.e., (value,node) pairs, rather than just nodes. For example, Fig. 8 shows an algorithm to compute the EV$^+$MDD (μ, r) encoding the minimum of the function encoded by the two EV$^+$MDDs (α, a) and (β, b), i.e., $v_{(\mu,r)}(\mathbf{i}) = \min\{v_{(\alpha,a)}(\mathbf{i}), v_{(\beta,b)}(\mathbf{i})\}$, or, in other words, $\mu + v_{(0,r)}(\mathbf{i}) = \min\{\alpha + v_{(0,a)}(\mathbf{i}), \beta + v_{(0,b)}(\mathbf{i})\}$, for all $\mathbf{i} \in \widehat{\mathcal{X}}$. The notation $p[i].child$ and $p[i].val$ is used to denote the node pointed by the i^{th} edge, and the associate value, respectively.

Finally, it should be stressed that the efficiency of computing $f \otimes g$ for some arbitrary operator $\otimes$ when the functions f and g are encoded by edge-valued decision diagrams depends on the relative properties of both the operator $\otimes$ and the operator $\odot$ used to combine the values along the paths of the decision diagram. For example, computing the elementwise minimum or sum of two functions encoded using EV$^+$MDDs is quite efficient, but computing their elementwise product is obviously more difficult.

3 Discrete State Models

We now describe the setting for the classes of system analysis we intend to perform symbolically, using decision diagrams. First we discuss logical analysis, then Markov analysis.

```
bdd  Or(bdd a, bdd b) is                                   • fully-reduced version
 local  bdd   r, r_0, r_1;
 local  level  k;
  1 if a = 0 or b = 1 then return b;
  2 if b = 0 or a = 1 then return a;
  3 if a = b then return a;
  4 if Cache contains entry ⟨ORcode, {a, b} : r⟩ then return r;
  5 if a.lvl < b.lvl then                    • compare the levels of nodes a and b
  6   k ← b.lvl;
  7   r_0 ← Or(a, b[0]);
  8   r_1 ← Or(a, b[1]);
  9 else if a.lvl > b.lvl then
 10   k ← a.lvl;
 11   r_0 ← Or(a[0], b);
 12   r_1 ← Or(a[1], b);
 13 else                                                  • a.lvl = b.lvl
 14   k ← a.lvl;
 15   r_0 ← Or(a[0], b[0]);
 16   r_1 ← Or(a[1], b[1]);
 17 r ← UniqueTableInsert(k, r_0, r_1);
 18 enter ⟨ORcode, {a, b} : r⟩ in Cache;
 19 return r;
```

```
bdd  RelProd(bdd x, bdd2 t) is                             • quasi-reduced version
 local  bdd   r, r_0, r_1;
  1 if x = 0 or t = 0 then return 0;
  2 if x = 1 and t = 1 then return 1;
  3 if Cache contains entry ⟨RELPRODcode, x, t : r⟩ then return r;
  4 r_0 ← Or(RelProd(x[0], t[0][0]), RelProd(x[1], t[1][0]));
  5 r_1 ← Or(RelProd(x[0], t[0][1]), RelProd(x[1], t[1][1]));
  6 r ← UniqueTableInsert(x.lvl, r_0, r_1);
  7 enter ⟨RELPRODcode, x, t : r⟩ in Cache;
```

Fig. 7. Examples of recursive algorithms on fully-reduced and quasi-reduced BDDs

3.1 Logic System Description and Analysis

We consider a discrete-state model $(\widehat{\mathcal{X}}, \mathcal{X}_{init}, \mathcal{T})$, where $\widehat{\mathcal{X}}$ is a finite set of states, $\mathcal{X}_{init} \subseteq \widehat{\mathcal{X}}$ is the set of initial states, and $\mathcal{T} \subseteq \widehat{\mathcal{X}} \times \widehat{\mathcal{X}}$ is a transition relation. We assume the *global* model state to be of the form $(x_L, ..., x_1)$, where, for $L \geq k \geq 1$, each *local* state variable x_k takes value from a set $\mathcal{X}_k = \{0, 1, ..., n_k - 1\}$, with $n_k > 0$. Thus, $\widehat{\mathcal{X}} = \mathcal{X}_L \times \cdots \times \mathcal{X}_1$ and we write $\mathcal{T}(i_L, ..., i_1, i'_K, ..., i'_1)$, or $\mathcal{T}(\mathbf{i}, \mathbf{i}')$, if the model can move from the *current state* $\mathbf{i}$ to a *next state* $\mathbf{i}'$ in one step.

The first step in system analysis is often the computation of the *reachable state space*. The goal is to find the set $\mathcal{X}_{reach}$ of states reachable from the initial set of states $\mathcal{X}_{init}$ according to the transition relation $\mathcal{T}$. Let $\mathbf{i} \rightarrow \mathbf{i}'$ mean that state $\mathbf{i}$ can reach state $\mathbf{i}'$ in one step, i.e., $(\mathbf{i}, \mathbf{i}') \in \mathcal{T}$, which we also write as $\mathbf{i}' \in \mathcal{T}(\mathbf{i})$ with a slight abuse of notation. Then, the reachable state space is

```
evmdd Min(level k, evmdd (α, a), evmdd (β, b)) is          • quasi-reduced version
 local  evmdd  (μ, r), r_0, ..., r_{n_k − 1}, (α', a'), (β', b');
  1 if α = ∞ then return (β, b);
  2 if β = ∞ then return (α, a);
  3 μ ← min(α, β);
  4 if k = 0 then return (μ, Ω);                  • the only node at level 0 is Ω
  5 if Cache contains entry ⟨MINcode, a, b, α − β : (γ, r)⟩ then return (γ + μ, r);
  6 for i = 0 to n_k − 1 do
  7   a' ← a[i].child;
  8   α' ← α − μ + a[i].val;
  9   b' ← b[i].child;
 10   β' ← β − μ + b[i].val;
 11   r_i ← Min(k−1, (α', a'), (β', b'));            • continue downstream
 12 r ← UniqueTableInsert(k, r_0, ..., r_{n_k − 1});
 13 enter ⟨MINcode, a, b, α − β : (μ, r)⟩ in Cache;
 14 return (μ, r);
```

Fig. 8. A recursive algorithms for EV$^+$MDDs

$$\mathcal{X}_{reach} = \{\mathbf{j} : \exists d > 0,\ \exists \mathbf{i}^{(1)} \rightarrow \mathbf{i}^{(2)} \rightarrow \cdots \rightarrow \mathbf{i}^{(d)}\ \wedge\ \mathbf{i}^{(1)} \in \mathcal{X}_{init}\ \wedge\ \mathbf{j} = \mathbf{i}^{(d)}\}.$$

Further logic analysis might involve searching for *deadlocks* or proving certain *liveness* properties. In general, such questions can be expressed in some *temporal logic*. Here, we assume the use of *computation tree logic (CTL)* [17,23]. This requires a *Kripke structure*, i.e., augmenting our discrete state model with a set of *atomic propositions* $\mathcal{A}$ and a *labeling function* $\mathcal{L} : \widehat{\mathcal{X}} \rightarrow 2^{\mathcal{A}}$ giving, for each state $\mathbf{i} \in \widehat{\mathcal{X}}$, the set of atomic propositions $\mathcal{L}(\mathbf{i}) \subseteq \mathcal{A}$ that hold in $\mathbf{i}$. Then, the syntax of CTL is as follows:

- if $a \in \mathcal{A}$, a is a *state formula*;
- if p and p' are state formulas, $\neg p$, $p \vee p'$, and $p \wedge p'$ are state formulas;
- if p and p' are state formulas, $\mathsf{X}p$, $\mathsf{F}p$, $\mathsf{G}p$, $p\mathsf{U}p'$, $p\mathsf{R}p'$ are *path formulas*;
- if q is a path formula, $\mathsf{E}q$ and $\mathsf{A}q$ are state formulas.

The semantic of CTL assigns a set of model states to each state formula p, thus, CTL operators must occur in pairs: a *path quantifier*, E or A, must always immediately precede a *temporal operator*, X, F, G, U, R. For brevity, we discuss only the semantics of the operator pairs EX, EU, and EG, since these are *complete*, meaning that they can be used to express any of the other seven CTL operators through complementation, conjunction, and disjunction:

- $\mathsf{AX}p = \neg\mathsf{EX}\neg p$,
- $\mathsf{EF}p = \mathsf{E}[true\ \mathsf{U}\ p]$,
- $\mathsf{E}[p\mathsf{R}q] = \neg\mathsf{A}[\neg p\mathsf{U}\neg q]$,
- $\mathsf{AF}p = \neg\mathsf{EG}\neg p$,
- $\mathsf{A}[p\ \mathsf{U}\ q] = \neg\mathsf{E}[\neg q\ \mathsf{U}\ \neg p\ \wedge\ \neg q] \wedge \neg\mathsf{EG}\neg q$,
- $\mathsf{A}[p\mathsf{R}q] = \neg\mathsf{E}[\neg p\mathsf{U}\neg q]$, and
- $\mathsf{AG}p = \neg\mathsf{EF}\neg p$,

where *true* is a predicate that holds in any state.

```
ExplicitBuildEX(p) is
 1 X ← {i ∈ X_reach : p ∈ labels(i)};        • initialize X with the states satisfying p
 2 while X ≠ ∅ do
 3   pick and remove a state i' from X;
 4   for each i ∈ T^{-1}(i') do               • state i can transition to state i'
 5     labels(i) ← labels(i) ∪ {EXp};
```

```
ExplicitBuildEU(p, q) is
 1 X ← {i ∈ X_reach : q ∈ labels(i)};        • initialize X with the states satisfying q
 2 for each i ∈ X do
 3   labels(i) ← labels(i) ∪ {E[pUq]};
 4 while X ≠ ∅ do
 5   pick and remove a state i' from X;
 6   for each i ∈ T^{-1}(i') do               • state i can transition to state i'
 7     if E[pUq] ∉ labels(i) and p ∈ labels(i) then
 8       labels(i) ← labels(i) ∪ {E[pUq]};
 9       X ← X ∪ {i};
```

```
ExplicitBuildEG(p) is
 1 X ← {i ∈ X_reach : p ∈ labels(i)};        • initialize X with the states satisfying p
 2 build the strongly connected components in the subgraph of T induced by X;
 3 Y ← {i : i is in one of these strongly connected component};
 4 for each i ∈ Y do
 5   labels(i) ← labels(i) ∪ {EGp};
 6 while Y ≠ ∅ do
 7   pick and remove a state i' from Y;
 8   for each i ∈ T^{-1}(i') do               • state i can transition to state i'
 9     if EGp ∉ labels(i) and p ∈ labels(i) then
10       labels(i) ← labels(i) ∪ {EGp};
11       Y ← Y ∪ {i};
```

Fig. 9. Explicit CTL model checking algorithms

Let $\mathcal{P}$ and $\mathcal{Q}$ be the sets of states satisfying two CTL formulas p and q, respectively. Then, the sets of states satisfying $\mathsf{EX}p$, $\mathsf{E}p\mathsf{U}q$, and $\mathsf{EG}p$ are:

$$\mathcal{X}_{\mathsf{EX}p} = \{\mathbf{i} : \exists \mathbf{i}', \mathbf{i} \rightarrow \mathbf{i}' \ \wedge \ \mathbf{i}' \in \mathcal{P}\},$$
$$\mathcal{X}_{\mathsf{E}p\mathsf{U}q} = \{\mathbf{i} : \exists d > 0, \exists \mathbf{i}^{(1)} \rightarrow \cdots \rightarrow \mathbf{i}^{(d)} \wedge \mathbf{i} = \mathbf{i}^{(1)} \wedge \mathbf{i}^{(d)} \in \mathcal{Q} \wedge \forall c, 1 \leq c < d, \ \mathbf{i}^{(c)} \in \mathcal{P}\},$$
$$\mathcal{X}_{\mathsf{EG}p} = \{\mathbf{i} : \forall d > 0, \exists \mathbf{i}^{(1)} \rightarrow \cdots \rightarrow \mathbf{i}^{(d)} \wedge \mathbf{i} = \mathbf{i}^{(1)} \wedge \forall c, 1 \leq c \leq d, \ \mathbf{i}^{(c)} \in \mathcal{P}\}.$$

Fig. 9 shows the pseudocode to compute the set of states satisfying $\mathsf{EX}p$, $\mathsf{E}p\mathsf{U}q$, and $\mathsf{EG}p$, or, more precisely, to "label" these states, assuming that the states satisfying the CTL formulas p and q have already been labeled. In other words, the labels corresponding to the subformulas of a CTL formulas are assigned first; of course, at the innermost level, the labeling corresponding to atomic propositions is just given by the labeling function $\mathcal{L}$ of the Kripke structure. Unlike state-space generation, these algorithms "walk backwards" in the transition relation, thus use the *inverse transition relation* $\mathcal{T}^{-1}$ instead of $\mathcal{T}$.

3.2 Markov System Description and Analysis

A discrete-state model or a Kripke structure can be extended by associating timing information to each state-to-state transition. If the time required for each transition is an exponentially distributed random variable independently sampled every time the state is entered and if all the transitions out of each state are "racing" concurrently, this defines an underlying *continuous-time Markov chain (CTMC)*. Formally, a CTMC is a stochastic process $\{X_t : t \in \mathbb{R}\}$ with a discrete state space $\mathcal{X}_{reach}$ satisfying the memoryless property, that is:

$$\forall r \geq 1, \forall t > t^{(r)} > \cdots > t^{(1)}, \forall \mathbf{i}, \mathbf{i}^{(1)}, ..., \mathbf{i}^{(r)} \in \mathcal{X}_{reach},$$

$$\Pr\{X_t = \mathbf{i} \mid X_{t^{(r)}} = \mathbf{i}^{(r)}, ..., X_{t^{(1)}} = \mathbf{i}^{(1)}\} = \Pr\{X_t = \mathbf{i} \mid X_{t^{(r)}} = \mathbf{i}^{(r)}\}.$$

We limit our discussion to *homogeneous* CTMCs, where the above probability depends on t and $t^{(r)}$ only through the difference $t - t^{(r)}$, i.e.,

$$\Pr\{X_t = \mathbf{i} \mid X_{t^{(r)}} = \mathbf{i}^{(r)}\} = \Pr\{X_{t-t^{(r)}} = \mathbf{i} \mid X_0 = \mathbf{i}^{(r)}\}.$$

Let $\boldsymbol{\pi}_t$ be the *probability vector* denoting the probability $\boldsymbol{\pi}_t[\mathbf{i}] = \Pr\{X_t = \mathbf{i}\}$ of each state $\mathbf{i} \in \mathcal{X}_{reach}$ at time $t \geq 0$. A homogeneous CTMC is then described by its *initial probability vector* $\boldsymbol{\pi}_0$, satisfying

$$\boldsymbol{\pi}_0[\mathbf{i}] > 0 \;\Leftrightarrow \mathbf{i} \in \mathcal{X}_{init}$$

and by its *transition rate matrix* $\mathbf{R}$, defined by

$$\forall \mathbf{i}, \mathbf{j} \in \mathcal{X}_{reach},\; \mathbf{R}[\mathbf{i}, \mathbf{j}] = \begin{cases} 0 & \text{if } \mathbf{i} = \mathbf{j} \\ \lim_{h \to 0} \Pr\{X_h = \mathbf{j} | X_0 = \mathbf{i}\}/h & \text{if } \mathbf{i} \neq \mathbf{j} \end{cases}$$

or its *infinitesimal generator matrix* $\mathbf{Q}$, defined by,

$$\forall \mathbf{i}, \mathbf{j} \in \mathcal{X}_{reach},\; \mathbf{Q}[\mathbf{i}, \mathbf{j}] = \begin{cases} -\sum_{\mathbf{l} \neq \mathbf{i}} \mathbf{R}[\mathbf{i}, \mathbf{l}] & \text{if } \mathbf{i} = \mathbf{j} \\ \mathbf{R}[\mathbf{i}, \mathbf{j}] & \text{if } \mathbf{i} \neq \mathbf{j}. \end{cases}$$

The short-term, or *transient*, behavior of the CTMC is found by computing the transient probability vector $\boldsymbol{\pi}_t$, which is the solution of the ordinary differential equation $d\boldsymbol{\pi}_t/dt = \boldsymbol{\pi}_t \mathbf{Q}$ with initial condition $\boldsymbol{\pi}_0$, thus it is given by the *matrix exponential* expression $\boldsymbol{\pi}_t = \boldsymbol{\pi}_0 e^{\mathbf{Q}t}$. The long-term, or *steady-state*, behavior is found by computing the steady-state probability vector $\boldsymbol{\pi} = lim_{t \to \infty} \boldsymbol{\pi}_t$; if the CTMC is *irreducible* (since we assume that $\mathcal{X}_{reach}$ is finite, this implies that $\mathcal{X}_{reach}$ is a single strongly-connected component), $\boldsymbol{\pi}$ is independent of $\boldsymbol{\pi}_0$ and is the unique solution of the homogeneous linear system $\boldsymbol{\pi}\mathbf{Q} = \mathbf{0}$ subject to $\sum_{\mathbf{i} \in \mathcal{X}_{reach}} \boldsymbol{\pi}[\mathbf{i}] = 1$. The probability vectors $\boldsymbol{\pi}$ and $\boldsymbol{\pi}_t$ are typically used to evaluate expected *instantaneous reward measures*. For example, a reward function $r : \mathcal{X}_{reach} \to \mathbb{R}$ specifies the rate at which a "reward" is generated in each state, and its expected value in steady state is computed as $\sum_{\mathbf{i} \in \mathcal{X}_{reach}} \boldsymbol{\pi}[\mathbf{i}] r(\mathbf{i})$.

```
real[n] Jacobi(real[n] π^(old), h, real[n,n] R) is
 local  real[n]  π^(new);
  1 repeat
  2   for j = 1 to |X_reach|
  3     π^(new)[j] ← h[j] · Σ_{i:R[i,j]>0} π^(old)[i] · R[i,j];
  4   π^(new) ← π^(new)/(π^(new) · 1);
  5   π^(old) ← π^(new);
  6 until "converged";
  7 return π^(new);
```

```
real[n] GaussSeidel(real[n] π, h, real[n,n] R) is
  1 repeat
  2   for j = 1 to |X_reach|
  3     π[j] ← h[j] · Σ_{i:R[i,j]>0} π[i] · R[i,j];
  4   π ← π/(π · 1);
  5 until "converged";
  6 return π;
```

```
real[n] Uniformization(real[n] π_0, real[n,n] P, real q,t, natural M) is
 local  real[n]  π_t;
  1 π_t ← 0;
  2 γ ← π_0;
  3 Poisson ← e^{-qt};
  4 for k = 1 to M do
  5   π_t ← π_t + γ · Poisson;
  6   γ ← γ · P;
  7   Poisson ← Poisson · q · t/k;
  8 return π_t;
```

Fig. 10. Numerical solution algorithms for CTMCs ($n = |\mathcal{X}_{reach}|$)

It is also possible to evaluate *accumulated reward measures* over a time interval $[t_1, t_2]$, in either the transient ($t_1 < t_2 < \infty$) or the long term ($t_1 < t_2 = \infty$). The numerical algorithms and the issues they raise are similar to those for instantaneous rewards discussed above, thus we omit them for brevity.

In practice, for exact steady-state analysis, the linear system $\boldsymbol{\pi}\mathbf{Q} = \mathbf{0}$ is solved using iterative methods such as Jacobi or Gauss-Seidel, since the matrix $\mathbf{Q}$ is typically extremely large and quite sparse. If $\mathbf{Q}$ is stored by storing matrix $\mathbf{R}$, in sparse row-wise or column-wise format, and the diagonal of $\mathbf{Q}$, as a full vector, the operations required for these iterative solution methods are vector-matrix multiplications, i.e., vector-column (of a matrix) dot products. In addition to $\mathbf{Q}$, the solution vector $\boldsymbol{\pi}$ must be stored, and most iterative methods (such as Jacobi) require one or more auxiliary vectors of the same dimension as $\boldsymbol{\pi}$, $|\mathcal{X}_{reach}|$. For extremely large CTMCs, these auxiliary vectors may impose excessive memory requirements.

If matrix $\mathbf{R}$ is stored in sparse column-wise format, and vector $\mathbf{h}$ contains the expected "holding time" for each state, where $\mathbf{h}[\mathbf{i}] = -1/\mathbf{Q}[\mathbf{i},\mathbf{i}]$, then the Jacobi

method can be written as in Fig. 10, where $\boldsymbol{\pi}^{(new)}$ is an auxiliary vector and the matrix $\mathbf{R}$ is accessed by columns, although the algorithm can be rewritten to access matrix $\mathbf{R}$ by rows instead. The method of Gauss-Seidel, also shown in Fig. 10, is similar to Jacobi, except the newly computed vector entries are used immediately; thus only the solution vector $\boldsymbol{\pi}$ is stored, with newly computed entries overwriting the old ones. The Gauss-Seidel method can also be rewritten to access $\mathbf{R}$ by rows, but this is not straightforward, requires an auxiliary vector, and adds more computational overhead [19].

For transient analysis, the *uniformization* method is most often used, as it is numerically stable and uses straightforward vector-matrix multiplications as its primary operations (Fig. 10). The idea is to *uniformize* the CTMC with a rate $q \geq \max_{\mathbf{i} \in \mathcal{X}_{reach}}\{|\mathbf{Q}[\mathbf{i},\mathbf{i}]|\}$ and obtain a discrete-time Markov chain with transition probability matrix $\mathbf{P} = \mathbf{Q}/q + \mathbf{I}$. The number of iterations M must be large enough to ensure that $\sum_{k=0}^{M} e^{-qt}(qt)^k/k!$ is very close to 1.

4 Putting It All Together: Structured System Analysis

We are now ready to show how the logical and numerical analysis algorithms of the previous section can be implemented *symbolically* using appropriate classes of decision diagrams. Most of these algorithms compute the *fixpoint* of some *functional*, i.e., a function transformer, where the fixpoint is a function encoded as a decision diagram.

4.1 Symbolic State Space Generation

State space generation is one of the simplest examples of symbolic fixpoint computation, and arguably the most important one. The reachable state space $\mathcal{X}_{reach}$ can be characterized as the *smallest* solution of the fixpoint equation

$$\mathcal{X} \subseteq \mathcal{X}_{init} \cup \mathcal{T}(\mathcal{X}).$$

Algorithm *Bfs* in Fig. 11 implements exactly this fixpoint computation, where sets and relations are stored using L-level and $2L$-level MDDs, respectively, i.e., node p encodes the set $\mathcal{X}_p$ having characteristic function v_p satisfying

$$v_p(i_L, ..., i_1) = 1 \Leftrightarrow (i_L, ..., i_1) \in \mathcal{X}_p.$$

The union of sets is simply implemented by applying the *Or* operator of Fig. 7 to their characteristic functions, and the computation of the states reachable in one step is implemented by using function *RelProd*, also from Fig. 7 (of course, the MDD version of these functions must be employed if MDDs are used instead of BDDs). Since it performs a breadth-first symbolic search, algorithm *Bfs* halts in exactly as many iterations as the maximum distance of any reachable state from the initial states.

Many high-level formalisms can be used to implicitly describe the state space by specifying the initial state or states, thus $\mathcal{X}_{init}$, and a rule to generate the

```
mdd Bfs(mdd X_init) is
 local  mdd  p;
  1 p ← X_init;
  2 repeat
  3   p ← Or(p, RelProd(p, T));
  4 until p does not change;
  5 return p;
```

```
mdd BfsChaining(mdd X_init) is
 local  mdd  p;
  1 p ← X_init;
  2 repeat
  3   for each e ∈ E do
  4     p ← Or(p, RelProd(p, T_e));
  5 until p does not change;
  6 return p;
```

```
mdd Saturation(mdd X_init) is                    • assumes quasi-reduced MDDs
  1 return Saturate(L, X_init);
```

```
mdd Saturate(level k, mdd p) is
 local  mdd  r, r_0, ..., r_{n_k-1};
  1 if p = 0 then return 0;
  2 if p = 1 then return 1;
  3 if Cache contains entry ⟨SATcode, p : r⟩ then return r;
  4 for i = to n_k − 1 do
  5   r_i ← Saturate(k − 1, p[i]);        • first, be sure that the children are saturated
  6 repeat
  7   choose e ∈ E_k, i, j ∈ X_k, such that r_i ≠ 0 and T_e[i][j] ≠ 0;
  8   r_j ← Or(r_j, RelProdSat(k − 1, r_i, T_e[i][j]));
  9 until r_0, ..., r_{n_k-1} do not change;
 10 r ← UniqueTableInsert(k, r_0, ..., r_{n_k-1});
 11 enter ⟨SATcode, p : r⟩ in Cache;
 12 return r;
```

```
mdd RelProdSat(level k, mdd q, mdd2 f) is
 local  mdd  r, r_0, ..., r_{n_k-1};
  1 if q = 0 or f = 0 then return 0;
  2 if Cache contains entry ⟨RELPRODSATcode, q, f : r⟩ then return r;
  3 for each i, j ∈ X_k such that q[i] ≠ 0 and f[i][j] ≠ 0 do
  4   r_j ← Or(r_j, RelProdSat(k − 1, q[i], f[i][j]));
  5 r ← Saturate(k, UniqueTableInsert(k, r_0, ..., r_{n_k-1}));
  6 enter ⟨RELPRODSATcode, q, f : r⟩ in Cache;
  7 return r.
```

Fig. 11. Symbolic breadth-first, chaining, and Saturation state-space generation

states reachable in one step form each state, thus $\mathcal{T}$. Most formalisms are not only "structured" in the sense that they define the model state through L variables $(x_L, \ldots, x_1)$, which is of course required for any symbolic approach, but also "asynchronous", in the sense that they *disjunctively partition* [8] the transition

relation $\mathcal{T}$ according to a set $\mathcal{E}$ of *events*. When $\mathcal{T}$ is expressed as $\mathcal{T} = \bigcup_{e\in\mathcal{E}} \mathcal{T}_e$, it is usually more efficient to deviate from a strict breadth-first approach. Algorithm *BfsChaining* in Fig. 11 implements a *chaining* approach [31] where the effect of applying each event is immediately accumulated as soon as it is computed. Chaining is based on the observation that the number of symbolic iterations might be reduced if the application of asynchronous events is compounded sequentially. While the search order is not strictly breadth-first anymore when chaining is used, the number of iterations of the repeat loop is at most as large as for breadth-first search, and usually much smaller. However, the efficiency of symbolic state-space generation is determined not just by the *number* of iterations but also by their *cost*, i.e., by the size of the MDDs involved. In practice, chaining has been shown to be quite effective in many models, but its effectiveness can be greatly affected by the order in which events are applied.

Much larger efficiency improvements, however, can be usually achieved with the *Saturation* algorithm [12,16]. Saturation is motivated by the observation that, in many distributed systems, *interleaving semantic* implies that multiple *events* may occur, each exhibiting a strong *locality*, i.e., affecting only a few state variables. We associate two sets of state variables with each event e:

$$\mathcal{V}_M(e)=\{x_k : \exists \mathbf{i}=(i_L,...,i_1), \exists \mathbf{i}'=(i'_L,...,i'_1), \mathbf{i}' \in \mathcal{T}_e(\mathbf{i}) \wedge i_k \neq i'_k\} \quad \text{and}$$
$$\mathcal{V}_D(e)=\{x_k : \exists \mathbf{i}=(i_L,...,i_1), \exists \mathbf{j}=(j_L,...,j_1), \forall h\neq k, i_h=j_h \wedge \mathcal{T}_e(\mathbf{i})\neq\emptyset \wedge \mathcal{T}_e(\mathbf{j})=\emptyset\},$$

the state variables that can be modified by e or that can disable e, respectively. Then, we let

$$Top(e) = \max\{k : x_k \in \mathcal{V}_M(e) \cup \mathcal{V}_D(e)\}$$

be the highest state variable, thus MDD level, affected by event e, and we partition the event set $\mathcal{E}$ into $\mathcal{E}_k = \{e : Top(e) = k\}$, for $L \geq k \geq 1$.

Saturation computes multiple "lightweight" nested fixpoints Starting from the quasi-reduced MDD encoding $\mathcal{X}_{init}$, Saturation begins by exhaustively applying events $e \in \mathcal{E}_1$ to each node p at level 1, until it is *saturated*, i.e., until $\bigcup_{e\in\mathcal{E}_1} \mathcal{T}_e(\mathcal{X}_p) \subseteq \mathcal{X}_p$. Then, it saturates nodes at level 2 by exhaustively applying to them all the events $e \in \mathcal{E}_2$, with the proviso that, if any new node at level 1 is created in the process, it is immediately saturated (by firing the events in $\mathcal{E}_1$ on it). The approach proceeds in bottom-up order, until the events in $\mathcal{E}_L$ have been applied exhaustively to the topmost node r. At this point, the MDD is saturated, $\bigcup_{e\in\mathcal{E}} \mathcal{T}_e(\mathcal{X}_r) \subseteq \mathcal{X}_r$, and r encodes the reachable state space $\mathcal{X}_{reach}$.

Experimentally, Saturation has been shown to be often several orders of magnitude more efficient than symbolic breadth-first iterations in both memory and time, when employed on asynchronous systems. Extensions of the Saturation algorithm have also been presented, where the size and composition of the local state spaces $\mathcal{X}_k$ is not known prior to generating the state space; rather, the local state spaces are built "on-the-fly" alongside the (global) reachable state space during the symbolic iterations [11,16].

4.2 Symbolic CTL Model Checking

Moving now to symbolic CTL model checking, we need MDD-based algorithms for the EX, EU, and EG operators. The first requires no fixpoint computation, as it can be computed in one step:

$$\mathcal{X}_{\mathsf{EX}p} = \mathcal{T}^{-1}(\mathcal{P})$$

The set of states satisfying EpUq can instead be characterized as the *smallest* solution of the fixpoint equation

$$\mathcal{X} \subseteq \mathcal{Q} \cup (\mathcal{P} \cap \mathcal{T}^{-1}(\mathcal{X})),$$

while the set of states satisfying EGp can be characterized as the *largest* solution of the fixpoint equation

$$\mathcal{X} \supseteq \mathcal{P} \cap \mathcal{T}^{-1}(\mathcal{X}).$$

Fig. 12 shows the pseudocode for a breadth-first-style symbolic implementation of these three operators. Again, all sets and relations are encoded using L-level and $2L$-level MDDs, respectively. Chaining and Saturation versions of the symbolic EU algorithm are possible when $\mathcal{T}$ is disjunctively partitioned. These are usually much more efficient in terms of both memory and time [15].

4.3 Symbolic Markov Analysis

Approaches that exploit the structured representation of the transition rate matrix $\mathbf{R}$ of a CTMC have been in use for over two decades, based on *Kronecker algebra* [18]. Formally, such approaches require $\mathbf{R}$ to be the submatrix corresponding to the reachable states of a matrix $\widehat{\mathbf{R}} \in \mathbb{R}^{|\widehat{\mathcal{X}}\times\widehat{\mathcal{X}}|}$ expressed as a sum of *Kronecker products*:

$$\mathbf{R} = \widehat{\mathbf{R}}[\mathcal{X}_{reach}, \mathcal{X}_{reach}] \quad \text{where} \quad \widehat{\mathbf{R}} = \sum_{e\in\mathcal{E}} \widehat{\mathbf{R}}_e \quad \text{and} \quad \widehat{\mathbf{R}}_e = \bigotimes_{L\geq k\geq 1} \mathbf{R}_{e,k}.$$

$\widehat{\mathbf{R}}_e$ is the transition rate matrix due to event e and can be decomposed as the Kronecker product of L matrices $\mathbf{R}_{e,k} \in \mathbb{R}^{|\widehat{\mathcal{X}}\times\widehat{\mathcal{X}}|}$, each expressing the contribution of the local state x_k to the rate of event e [20,30]. Recall that the Kronecker product of two matrices $\mathbf{A} \in \mathbb{R}^{n_r\times n_c}$ and $\mathbf{B} \in \mathbb{R}^{m_r\times m_c}$ is a matrix $\mathbf{A}\otimes\mathbf{B} \in \mathbb{R}^{n_r\cdot m_r\times n_c\cdot m_c}$ satisfying

$$\forall i_a\in\{0,...,n_r-1\}, \forall j_a\in\{0,...,n_c-1\}, \forall i_b\in\{0,...,m_r-1\}, \forall j_b\in\{0,...,m_c-1\},$$

$$(\mathbf{A}\otimes\mathbf{B})[i_a\cdot m_r + i_b, j_a\cdot m_c + j_b] = \mathbf{A}[i_a, j_a]\cdot\mathbf{B}[i_b, j_b].$$

The algorithms for the numerical solution of CTMCs shown in Fig. 10 essentially perform a sequence of vector-matrix multiplications at their core. Thus, the efficient computation of $\mathbf{y} \leftarrow \mathbf{x}\cdot\mathbf{A}$ or $\mathbf{y} \leftarrow \mathbf{y}+\mathbf{x}\cdot\mathbf{A}$, when $\mathbf{A}$, or, rather, $\widehat{\mathbf{A}}$, is encoded as a Kronecker product $\bigotimes_{L\geq k\geq 1}\mathbf{A}_k$, has been studied at length [6]. The

```
mdd SymbolicBuildEX(mdd P, mdd2 T^-1) is
 local mdd X;
  1 X ← RelProd(P, T^-1);      • perform one backward step in the transition relation
  2 return X;
```

```
mdd SymbolicBuildEU(mdd P, mdd Q, mdd2 T^-1) is
 local mdd O, X, Y, Z;
  1 X ← Q;                     • initialize the currently known result with the states satisfying q
  2 repeat
  3   O ← X;                                              • save the old set of states
  4   Y ← RelProd(X, T^-1);    • perform one backward step in the transition relation
  5   Z ← And(Y, P);           • perform set intersection to discard states not satisfying p
  6   X ← Or(Z, X);                                       • add to the currently known result
  7 until O = X;
  8 return X;
```

```
mdd SymbolicBuildEG(mdd P, mdd2 T^-1) is
 local mdd O, X, Y;
  1 X ← P;                                    • initialize X with the states satisfying p
  2 repeat
  3   O ← X;                                              • save the old set of states
  4   Y ← RelProd(X, T^-1);    • perform one backward step in the transition relation
  5   X ← And(X, Y);
  6 until O = X;
  7 return X;
```

Fig. 12. Symbolic CTL model checking algorithms for the EX, EU, and EG operators

well-known *shuffle algorithm* [18,20] or other algorithms presented in [6] can be employed to compute this product, but their efficiency stems from two properties common to all algorithms for decision diagram manipulation. First, the object being encoded ($\mathbf{A}$, in this case) can be structured into L levels (the matrices $\mathbf{A}_k$, in this case). Second, some kind of caching is used to avoid recomputing the result of operations already performed.

One disadvantage of earlier Kronecker-based approaches for the steady-state or transient solution of a CTMC was that the probability vector $\boldsymbol{\pi} \in \mathbb{R}^{|\mathcal{X}_{reach}|}$ was actually stored using a possibly much larger probability vector $\widehat{\boldsymbol{\pi}} \in \mathbb{R}^{|\widehat{\mathcal{X}}|}$, to simplify state indexing. Correctness was achieved by ensuring that $\widehat{\boldsymbol{\pi}}[\mathbf{i}] = 0$ for all $\mathbf{i} \in \widehat{\mathcal{X}} \setminus \mathcal{X}_{reach}$, but the additional computational and, especially, memory overhead was substantial. Decision diagrams helped first by providing an efficient encoding for state indices. Given the MDD for $\mathcal{X}_{reach} \subseteq \widehat{\mathcal{X}}$, an EV$^+$MDD encoding the *lexicographic state indexing* function $\psi : \widehat{\mathcal{X}} \to \mathbb{N} \cup \{\infty\}$ can be easily built. Its definition is such that $\psi(\mathbf{i})$ is the number of reachable states preceding $\mathbf{i}$ in lexicographic order, if $\mathbf{i} \in \mathcal{X}_{reach}$, and is ∞ otherwise. This EV$^+$MDD has exactly the same number of nodes and edges as the MDD encoding $\mathcal{X}_{reach}$. For example, Fig. 13 shows a reachable state space $\mathcal{X}_{reach}$, the MDD encoding it, and the EV$^+$MDD encoding ψ. To compute the index of a state, sum the

$$\mathcal{X}_{reach} = \left\{ \begin{matrix} 0 \\ 2 \\ 1 \\ 0 \end{matrix}, \begin{matrix} 1 \\ 0 \\ 0 \\ 0 \end{matrix}, \begin{matrix} 1 \\ 0 \\ 1 \\ 0 \end{matrix}, \begin{matrix} 1 \\ 1 \\ 0 \\ 0 \end{matrix}, \begin{matrix} 1 \\ 1 \\ 1 \\ 0 \end{matrix}, \begin{matrix} 1 \\ 2 \\ 1 \\ 0 \end{matrix}, \begin{matrix} 2 \\ 0 \\ 0 \\ 0 \end{matrix}, \begin{matrix} 2 \\ 0 \\ 1 \\ 0 \end{matrix}, \begin{matrix} 2 \\ 1 \\ 0 \\ 0 \end{matrix}, \begin{matrix} 2 \\ 1 \\ 1 \\ 0 \end{matrix}, \begin{matrix} 2 \\ 2 \\ 1 \\ 0 \end{matrix}, \begin{matrix} 3 \\ 0 \\ 1 \\ 0 \end{matrix}, \begin{matrix} 3 \\ 1 \\ 1 \\ 0 \end{matrix}, \begin{matrix} 3 \\ 2 \\ 0 \\ 0 \end{matrix}, \begin{matrix} 2 \\ 0 \\ 0 \\ 1 \end{matrix}, \begin{matrix} 3 \\ 2 \\ 0 \\ 2 \end{matrix}, \begin{matrix} 3 \\ 2 \\ 1 \\ 0 \end{matrix}, \begin{matrix} 3 \\ 2 \\ 1 \\ 1 \end{matrix}, \begin{matrix} 3 \\ 2 \\ 1 \\ 2 \end{matrix} \right\}$$

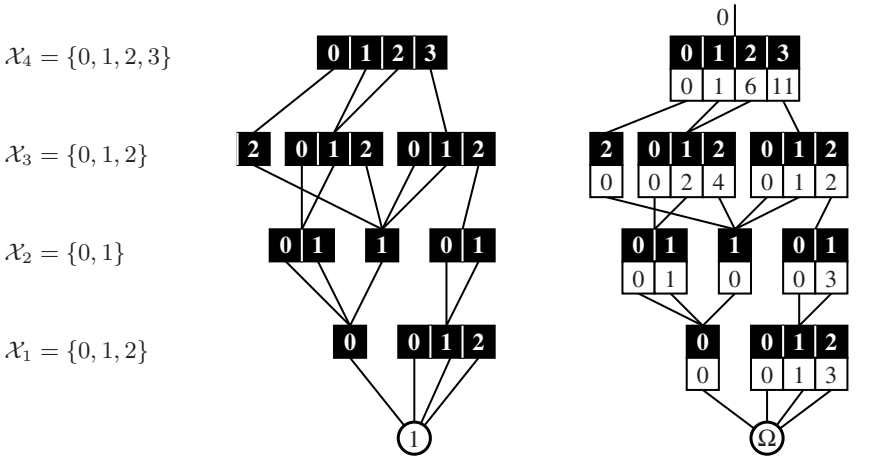


Fig. 13. Encoding the lexicographic state indexing function ψ using EV$^+$MDDs

values found on the corresponding path: $\psi(2, 1, 1, 0) = 6 + 2 + 1 + 0 = 9$; if a state is unreachable, the path is not complete: $\psi(0, 2, 0, 0) = 0 + 0 + \infty = \infty$. With this encoding of ψ, the probability vector $\boldsymbol{\pi}$ can be stored in a full array of size $|\mathcal{X}_{reach}|$, instead of $|\widehat{\mathcal{X}}|$, with little indexing overhead, making the Kronecker approach much more memory efficient.

However, at their best, the Kronecker algorithms that have been presented in the literature can barely match the efficiency of good decision diagram algorithms. Thus, we limit our discussion to the latter, and, in particular, to MxDs, which were indeed defined as a more general and efficient alternative to a Kronecker encoding. In this context, an MxD can be seen as a generalization of a Kronecker encoding, as both multiply L elements, corresponding to the L local states, to obtain an entry of the encoded matrix, and both exploit the likely occurrence of an identity matrix at any level to avoid useless "multiplications by 1". In addition, MxD can encode arbitrary matrices $\widehat{\mathbf{R}}_e$, even those that are cannot be expressed as a Kronecker product of L local matrices, and can reflect the actual reachability of states. This means that, given the MDD encoding $\mathcal{X}_{reach}$ and an MxD encoding a matrix $\widehat{\mathbf{R}}$ such that $\widehat{\mathbf{R}}[\mathcal{X}_{reach}, \mathcal{X}_{reach}] = \mathbf{R}$, we can enforce the fact that the entries of $\widehat{\mathbf{R}}$ should be 0 for any unreachable row $\mathbf{i} \in \widehat{\mathcal{X}} \setminus \mathcal{X}_{reach}$. Alternatively, these *spurious* nonzero entries can be dealt with explicitly, by testing for reachability, when using Gauss-Seidel iterations, which are best implemented with access-by-column to $\mathbf{R}$ (with the Kronecker approach, only this second explicit filtering of the spurious entries is possible).

We now discuss algorithm *VectorMatrixMult* of Fig. 14, which multiplies a *full* real vector $\mathbf{x}$ of size $|\mathcal{X}_{reach}|$ by the submatrix $\widehat{\mathbf{A}}[\mathcal{X}_{reach}, \mathcal{X}_{reach}]$, where $\widehat{\mathbf{A}}$ is

```
real[n] VectorMatrixMult(real[n] x, mxd_node A, evmdd_node ψ) is
 local  natural  s;                                          • state index in x
 local  real[n]  y;
 local  sparse_real  c;
  1  s ← 0;
  2  for each j = (j_L, ..., j_1) ∈ X_reach in lexicographic order do   • s = ψ(j)
  3    c ← GetCol(L, A, ψ, j_L, ..., j_1);   • build column j of A using sparse storage
  4    y[s] ← ElementWiseMult(x, c);   • x uses full storage, c uses sparse storage
  5    s ← s + 1;
  6  return y;
```

```
sparse_real GetCol(level k, mxd_node M, evmdd_node φ, natural j_k, ..., j_1) is
 local  sparse_real  c, d;
  1  if k = 0 then return [1];          • a vector of size one, with its entry set to 1
  2  if Cache contains entry ⟨COLcode, M, φ, j_k, ..., j_1 : c⟩ then return c;
  3  c ← 0;                                  • initialize the results to all zero entries
  4  for each i_k ∈ X_k such that M[i_k, j_k].val ≠ 0 and φ[i_k].val ≠ ∞ do
  5    d ← GetCol(k − 1, M[i_k, j_k].child, φ[i_k].child, j_{k−1}, ..., j_1);
  6    for each i such that d[i] ≠ 0 do
  7      c[i + φ[i_k].val] ← c[i + φ[i_k].val] + M[i_k, j_k].val · d[i];
  8  enter ⟨COLcode, M, φ, j_k, ..., j_1 : c⟩ in Cache;
  9  return c;
```

Fig. 14. MxD-based vector-matrix multiplication algorithm ($n = |\mathcal{X}_{reach}|$)

encoded by an MxD rooted at node A. For simplicity, we ignore the value ρ of the incoming dangling edge, i.e., we assume that $\rho = 1$. In practice, we could enforce this assumption by allowing the root node A, and only it, to be unnormalized, so that its entries are multiplied by ρ. Also for simplicity, we assume that the MxD is quasi-reduced, thus $A.lvl = L$. The correctness of this algorithm does not depend on the absence of spurious entries in $\widehat{\mathbf{A}}$, since the lexicographic state indexing function, encoded by an EV$^+$MDD with root edge $(0, \psi)$, is used to select only the rows corresponding to reachable states.

Algorithm *VectorMatrixMult* operates by building the column of $\widehat{\mathbf{A}}$ corresponding to each reachable state $\mathbf{j}$. As such column is usually very sparse, it is stored in a sparse data structure, not as a full array. The key procedure is *GetCol* [29], which recursively builds the required column, filtering out entries corresponding to unreachable rows. Once the column $\mathbf{c}$ is returned, *VectorMatrixMult* can perform an efficient multiplication of $\mathbf{x}^T \cdot \mathbf{c}$, by simply examining the nonzero entries of $\mathbf{c}$ and using direct access to the corresponding entries of $\mathbf{x}$. Note that, since the columns $\mathbf{j}$ are built in lexicographic order, the state index s can be simply incremented for each new column.

The algorithm just presented is considered the current state-of-the-art, but it nevertheless shares an important limitation with all *hybrid* approaches (including Kronecker-based ones): the vector $\mathbf{x}$, thus the probability vector $\boldsymbol{\pi}$, uses full storage, thus requires $O(|\mathcal{X}_{reach}|)$ memory. Sparse storage does not help (since none of its entries is zero if the CTMC is ergodic) and symbolic storage usually requires even more memory, since the MTMDD or the EV$^+$MDD (or its

multiplicative analogue where edge values are multiplied along the path) end up being close to a tree with as many leaves as reachable states. Strictly symbolic approaches where not only $\mathbf{R}$ but also $\boldsymbol{\pi}$ is stored using decision diagrams have been successful so far only when many states share the same probability, usually a clear indication that the CTMC exhibits strong symmetries, and is thus *lumpable* [25].

5 Conclusion and Future Areas of Research

Major advances in our capability of analyzing large and complex systems have been already achieved through the use of decision diagram techniques. Nevertheless, several research challenges still lie ahead.

Variable ordering heuristics. It is well known that the ordering of the L state variables can exponentially affect the size of the decision diagrams, thus the efficiency of their manipulation. Unfortunately, finding an optimal variable order is known to be a hard problem [2], thus, heuristics for either the static ordering (at the beginning of the analysis, prior to building any decision diagram) or the dynamic ordering (during the analysis, if the decision diagrams become too large) of state variables have been proposed with varying degree of success [22].

Most approaches have been targeted at BDDs and breadth-first iterations. We have begun considering the special requirements of Saturation and MDDs in [32,10], but much more work is required to explore heuristics for arbitrary classes of decision diagrams and their manipulations.

Strictly symbolic numerical CTMC solution. As mentioned, hybrid solution approaches for CTMCs are probably as efficient as they can be, at least in a general setting, i.e., unless the model belongs to a special class whose properties can be exploited to gain some efficiency on a case-by-case basis. A strictly symbolic approach, possibly allowing for a controlled level of approximation, might then be a very valuable contribution. Along these lines, an approach that uses exact symbolic representations of $\mathcal{X}_{reach}$ and $\mathbf{R}$ but an approximate representation for $\boldsymbol{\pi}$ has been proposed [28], but much more work is required.

References

1. R. I. Bahar, E. A. Frohm, C. M. Gaona, G. D. Hachtel, E. Macii, A. Pardo, and F. Somenzi. Algebraic decision diagrams and their applications. *Formal Methods in System Design*, 10(2/3):171–206, Apr. 1997.
2. B. Bollig and I. Wegener. Improving the variable ordering of OBDDs is NP-complete. *IEEE Trans. Comp.*, 45(9):993–1002, Sept. 1996.
3. R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Trans. Comp.*, 35(8):677–691, Aug. 1986.
4. R. E. Bryant. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Comp. Surv.*, 24(3):293–318, 1992.

5. P. Buchholz. Structured analysis approaches for large Markov chains. *Applied Numerical Mathematics*, 31(4):375–404, 1999.
6. P. Buchholz, G. Ciardo, S. Donatelli, and P. Kemper. Complexity of memory-efficient Kronecker operations with applications to the solution of Markov models. *INFORMS J. Comp.*, 12(3):203–222, 2000.
7. P. Buchholz, J. P. Katoen, P. Kemper, and C. Tepper. Model-checking large structured Markov chains. *J. Logic & Algebraic Progr.*, 56(1/2):69–97, 2003.
8. J. R. Burch, E. M. Clarke, and D. E. Long. Symbolic model checking with partitioned transition relations. In A. Halaas and P.B. Denyer, editors, *Int. Conference on Very Large Scale Integration*, pages 49–58, Edinburgh, Scotland, Aug. 1991. IFIP Transactions, North-Holland.
9. G. Ciardo, R. L. Jones, A. S. Miner, and R. Siminiceanu. Logical and stochastic modeling with SMART. *Perf. Eval.*, 63:578–608, 2006.
10. G. Ciardo, G. Lüttgen, and A. J. Yu. Improving static variable orders via invariants. In *Proc. 28th International Conference on Application and Theory of Petri nets and Other Models of Concurrency (ICATPN)*, Siedlce, Poland, June 2007. Springer-Verlag. To appear.
11. G. Ciardo, R. Marmorstein, and R. Siminiceanu. Saturation unbound. In H. Garavel and J. Hatcliff, editors, *Proc. Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, LNCS 2619, pages 379–393, Warsaw, Poland, Apr. 2003. Springer-Verlag.
12. G. Ciardo, R. Marmorstein, and R. Siminiceanu. The saturation algorithm for symbolic state space exploration. *Software Tools for Technology Transfer*, 8(1):4–25, Feb. 2006.
13. G. Ciardo and A. S. Miner. A data structure for the efficient Kronecker solution of GSPNs. In P. Buchholz, editor, *Proc. 8th Int. Workshop on Petri Nets and Performance Models (PNPM'99)*, pages 22–31, Zaragoza, Spain, Sept. 1999. IEEE Comp. Soc. Press.
14. G. Ciardo and R. Siminiceanu. Using edge-valued decision diagrams for symbolic generation of shortest paths. In M. D. Aagaard and J. W. O'Leary, editors, *Proc. Fourth International Conference on Formal Methods in Computer-Aided Design (FMCAD)*, LNCS 2517, pages 256–273, Portland, OR, USA, Nov. 2002. Springer-Verlag.
15. G. Ciardo and R. Siminiceanu. Structural symbolic CTL model checking of asynchronous systems. In W. Hunt, Jr. and F. Somenzi, editors, *Computer Aided Verification (CAV'03)*, LNCS 2725, pages 40–53, Boulder, CO, USA, July 2003. Springer-Verlag.
16. G. Ciardo and A. J. Yu. Saturation-based symbolic reachability analysis using conjunctive and disjunctive partitioning. In D. Borrione and W. Paul, editors, *Proc. CHARME*, LNCS 3725, pages 146–161, Saarbrücken, Germany, Oct. 2005. Springer-Verlag.
17. E. M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Proc. IBM Workshop on Logics of Programs*, LNCS 131, pages 52–71, London, UK, 1981. Springer-Verlag.
18. M. Davio. Kronecker products and shuffle algebra. *IEEE Trans. Comp.*, C-30:116–125, Feb. 1981.
19. D. D. Deavours and W. H. Sanders. "On-the-fly" solution techniques for stochastic Petri nets and extensions. In *Proc. 7th Int. Workshop on Petri Nets and Performance Models (PNPM'97)*, pages 132–141, Saint-Malo, France, June 1997. IEEE Comp. Soc. Press.

20. P. Fernandes, B. Plateau, and W. J. Stewart. Efficient descriptor-vector multiplication in stochastic automata networks. *J. ACM*, 45(3):381–414, 1998.
21. M. Fujita, P. C. McGeer, , and J. C.-Y. Yang. Multi-terminal binary decision diagrams: an efficient data structure for matrix representation. *Formal Methods in System Design*, 10:149–169, 1997.
22. O. Grumberg, S. Livne, and S. Markovitch. Learning to order BDD variables in verification. *J. Art. Int. Res.*, 18:83–116, 2003.
23. K. L. McMillan. *Symbolic Model Checking*. Kluwer, 1993.
24. T. Kam, T. Villa, R. Brayton, and A. Sangiovanni-Vincentelli. Multi-valued decision diagrams: theory and applications. *Multiple-Valued Logic*, 4(1–2):9–62, 1998.
25. J. G. Kemeny and J. L. Snell. *Finite Markov Chains*. D. Van Nostrand-Reinhold, New York, NY, 1960.
26. Y.-T. Lai and S. Sastry. Edge-valued binary decision diagrams for multi-level hierarchical verification. In *Proceedings of the 29th Conference on Design Automation*, pages 608–613, Los Alamitos, CA, USA, June 1992. IEEE Computer Society Press.
27. A. S. Miner. Efficient solution of GSPNs using canonical matrix diagrams. In R. German and B. Haverkort, editors, *Proc. 9th Int. Workshop on Petri Nets and Performance Models (PNPM'01)*, pages 101–110, Aachen, Germany, Sept. 2001. IEEE Comp. Soc. Press.
28. A. S. Miner, G. Ciardo, and S. Donatelli. Using the exact state space of a Markov model to compute approximate stationary measures. In J. Kurose and P. Nain, editors, *Proc. ACM SIGMETRICS*, pages 207–216, Santa Clara, CA, USA, June 2000. ACM Press.
29. A. S. Miner and D. Parker. Symbolic representations and analysis of large state spaces. In C. Baier, B. R. Haverkort, H. Hermanns, J.-P. Katoen, and M. Siegle, editors, *Validation of Stochastic Systems*, LNCS 2925, pages 296–338. Springer-Verlag, 2004.
30. B. Plateau. On the stochastic structure of parallelism and synchronisation models for distributed algorithms. In *Proc. ACM SIGMETRICS*, pages 147–153, Austin, TX, USA, May 1985.
31. O. Roig, J. Cortadella, and E. Pastor. Verification of asynchronous circuits by BDD-based model checking of Petri nets. In G. De Michelis and M. Diaz, editors, *Proc. 16th Int. Conf. on Applications and Theory of Petri Nets*, LNCS 935, pages 374–391, Turin, Italy, June 1995. Springer-Verlag.
32. R. Siminiceanu and G. Ciardo. New metrics for static variable ordering in decision diagrams. In H. Hermanns and J. Palsberg, editors, *Proc. Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, LNCS 3920, pages 90–104, Vienna, Austria, Mar. 2006. Springer-Verlag.

Introduction to Software Performance Engineering: Origins and Outstanding Problems

Connie U. Smith

Performance Engineering Services
P.O. Box 2640
Santa Fe, NM 87504
www.spe-ed.com

Abstract. This chapter first reviews the origins of Software Performance Engineering (SPE). It provides an overview and an extensive bibliography of the early research. It then covers the fundamental elements of SPE: the data required, the software performance models and the SPE process. It illustrates how to apply the modeling and analysis techniques with a case study. It concludes with a review of the current status and outstanding problems.

Keywords: Software Performance Engineering, SPE, Performance prediction, Performance models, Performance patterns, Performance Antipattterns, SPE Process, Queueing Network Models.

1 Introduction

Software performance engineering (SPE) is a systematic, quantitative approach to constructing software systems that meet performance requirements. In this chapter, *performance* refers to the response time or throughput as seen by the users. For real-time, or *reactive* systems, it is the time required to respond to events or the number of events processed in a time interval. Reactive systems must meet performance requirements to be correct. Other software has less stringent requirements, but responsiveness limits the amount of work processed, so it determines a system's effectiveness and the productivity of its users.

SPE provides an engineering approach to performance, avoiding the extremes of performance-driven development and "fix-it-later." SPE uses model predictions to evaluate trade-offs in software functions, hardware size, quality of results, and resource requirements.

SPE is a software-oriented approach: it focuses on architecture, design, and implementation choices. The models assist developers in controlling resource requirements by selecting architecture and design alternatives with acceptable performance characteristics. They aid in tracking performance throughout the development process and prevent problems from surfacing late in the life cycle.

SPE also provides principles, patterns, and antipatterns for creating responsive software, specifications for the data required for evaluation, procedures for obtaining performance specifications, and guidelines for the types of evaluation to be conducted

M. Bernardo and J. Hillston (Eds.): SFM 2007, LNCS 4486, pp. 395–428, 2007.

at each development stage. It incorporates models for representing and predicting performance as well as a set of analysis techniques [105, 122].

SPE techniques provide the following information about the new system:

- Refinement and clarification of the performance requirements
- Predictions of performance with precision matching the software knowledge available in the early development stage and the quality of resource usage estimates available at that time
- Estimates of the sensitivity of the predictions to the accuracy of the resource usage estimates and workload intensity
- Understanding of the quantitative impact of design alternatives, that is the effect of system changes on performance
- Scalability of the architecture and design: the effect of future growth on performance
- Identification of critical parts of the design
- Identification of assumptions that, if violated, could change the assessment
- Assistance for budgeting resource demands for parts of the design
- Assistance in designing performance tests

This chapter first covers the evolution of SPE then it gives an overview of the SPE process and techniques. It describes the general principles for performance-oriented design, the performance patterns and antipatterns, and the quantitative techniques for predicting and analyzing performance. A case study illustrates the modeling and analysis techniques. Finally, the conclusion reviews the status and future of SPE.

2 The Evolution of SPE

Performance was typically considered in the early years of computing. Knuth's early work focused on efficient data structures, algorithms, sorting and searching [60, 61]. The space and time required by programs had to be carefully managed to fit them on small machines. The hardware grew but, rather than eliminating performance problems, it made larger, more complex software feasible and programs grew into systems of programs. Software systems with strict performance requirements, such as flight control systems and other embedded systems used detailed simulation models to assess performance. Consequently creating and solving them was time-consuming, and updating the models to reflect the current state of evolving software systems was problematic. Thus, the labor-intensive modeling and assessment were cost-effective only for systems with strict performance requirements.

Authors proposed performance-oriented development approaches [13, 41, 85, 97] but most developers of non-reactive systems adopted the "fix-it-later" methodology. It advocated concentrating on software correctness, deferring performance considerations to the integration testing phase and (if performance problems were detected then) procuring additional hardware or "tuning" the software to correct them. Fix-it-later was acceptable in the 1970s, but in the 1980s the demand for computer resources increased dramatically. System complexity increased while the proportional number of developers with performance expertise decreased. This, combined with a

directive to ignore performance, made the resulting performance disasters a self-fulfilling prophecy. Many of the disasters could not be corrected with hardware – platforms with the required power did not exist. Neither could they be corrected with tuning – corrections required major design changes, and thus reimplementation. Meanwhile, technical advances led to the SPE alternative.

2.1 Modeling Foundations

In 1971, Buzen proposed modeling systems with queueing network models and published efficient algorithms for solving some important models [26]. In 1975, Baskett, et. al., defined a class of models that satisfy separability constraints and thus have efficient analytic solutions [7]. The models are an abstraction of the computer systems they model, so they are easier to create than general-purpose simulation models. Because they are solved analytically, they can be used interactively. Since then, many advances have been made in modeling computer systems with queuing networks, faster solution techniques, and accurate approximation techniques [44, 55, 72].

Queueing network models are commonly used in capacity planning to model computer systems. A capacity planning model is constructed from specifications for the computer system configuration and measurements of resource requirements for each of the workloads modeled. The model is solved and the resulting performance metrics (response time, throughput, resource utilization, etc.) are compared to measured performance. The model is calibrated to the computer system. Then, it is used to study the effect of increases in workload and resource demands, and of configuration changes.

Initially, queueing network models were used primarily for capacity planning. For SPE they were sometimes used for feasibility analysis: request arrivals and resource requirements were estimated and the results assessed. More precise models were infeasible because the software could not be measured until it was implemented.

The second SPE modeling breakthrough was the introduction of analytical models for software [10, 19, 20, 22, 91, 106, 129]. With them, software execution is modeled, estimates of resource requirements are made, and performance metrics are calculated. Software execution models yield an approximate value for best, worst, or average resource requirements. They provide an estimate for response time; they can detect response time problems, but because they do not model resource contention they do not yield precise values for predicted response time.

The third SPE modeling breakthrough was combining the analytic software models with the queueing-network system models to more precisely model execution characteristics [18]; [98, 107]. Combined models more precisely model the execution. They also show the effect of new software on existing work and on resource utilization. They identify computer device bottlenecks and the parts of the new software with high use of bottleneck devices.

By 1980, the modeling power was established and modeling tools were available [18]; [54]; [84]. Many new tools are now available. Thus, it became cost-effective to model large software systems early in their development.

2.2 SPE Methods

Early experience with a large system confirmed that sufficient data could be collected early in development to predict performance bottlenecks [108]. Unfortunately, despite the predictions, the system design was not modified to remove them and upon implementation (approximately one year later) performance in those areas was a serious problem, as predicted. The modeling problems were resolved, but that was not enough to prevent problems.

SPE methods were proposed [99] and later updated [105, 122]. Key parts of the SPE process are methods for collecting data early in software development, and critical success factors to ensure SPE success. The methods also address compatibility with software engineering methods, what is done, when, by whom, and other organizational issues.

2.3 SPE Development

The 1980s and 90s brought advances in all facets of SPE. Software model advances were proposed by several authors [11, 21, 33, 83, 90, 100, 105, 111]. Martin [70] proposed data-action graphs as a representation that facilitates transformation between performance models and various software design notations. Opdahl and Sølvberg [78] integrated information system models and performance models with extended specifications. Brataas applied SPE modeling techniques to organizational workflow analysis in [23].

Rolia [86] extended the SPE models and methods to address systems of cooperating processes in a distributed and multicomputer environment. Woodside [135, 136] proposed stochastic rendezvous networks to evaluate performance of Ada systems and Woodside and coworkers incorporated the analysis techniques into a software engineering tool [24, 137]. Opdahl [77] described SPE tools interfaced with the PPP CASE tool and the IMSE environment for performance modeling – both were part of the Esprit research initiative. Lor and Berry [69] automatically generated SARA models from an arbitrary requirements specification language using a knowledge-based Design Assistant.

SPE models were extended and applied to Client/Server systems [72, 125], to distributed systems [48, 95, 117], and web applications [29, 46, 71, 94, 120]

Newer tools that incorporate features to support SPE modeling are reported by numerous authors [5, 9, 63, 74, 75, 81, 114, 116].

Extensive advances have been made in computer system performance modeling techniques. A complete list of references is beyond the scope of this chapter (for more information see other chapters in this book).

Bentley [15] proposed a set of rules for writing efficient programs. A set of formal, general principles for performance-oriented design is reported by Smith [102, 103, 105]. These principles were extended to software performance patterns and antipatterns [119, 122]. Software architects who are experts in formulating requirements and designs, and developers who are experts in data structure and algorithm selection, use intuition to develop their systems. The rules, principles, and patterns formalize that expert knowledge. Thus, the expert knowledge developed

through years of experience, can be easily transferred to software developers with less experience via the performance patterns and antipatterns.

Numerous authors relay experience with SPE [1, 2, 4, 12, 14, 37, 38, 79, 101, 104] The *Proceedings of the Computer Measurement Group Conferences* contain SPE experience papers each year (see [28]).

SPE has also been adapted to embedded real-time systems. When these systems must respond to events within a specified time interval, they are called *reactive systems*. Much of the real-time systems work examines resource allocation or schedulability. Other work expresses timing requirements with assertions and proves that requirements are met. Both of those approaches are outside the scope of the software-oriented SPE approach. Many other authors propose methods for performance-oriented design but do not use SPEs quantitative approach. Levi and Agrawala [68] proposed a comprehensive system for creating real-time systems – it encompasses object-oriented descriptions, implementations, techniques, and an operating system for scheduling and execution of the resulting software. Howes [50] prescribed principles for developing efficient reactive systems. Williams and Smith [112] formalized the SPE process for real-time systems. Sholl and Kim [96] adapted the computation-structure approach to real-time systems, and LeMer [67] described a methodology and tool. Chang and coworkers [27] used Petri net model extensions to evaluate real-time systems.

The SPE process, models, and tools have also been extended to object-oriented systems [58, 114, 115, 122]. Performance measurements have been used to generate performance models [51, 52, 88]. A workshop has been created to specifically address software and performance issues [140].

Models for new types of systems are created as soon as new technology emerges, such as Web Services and Service Oriented Architecture. They can be found in many publications including [28, 82, 140].

After highlighting the key elements of SPE in the next three sections, the final section mentions additional related work and discusses outstanding problems.

2.4 The SPE Process

The SPE process prescribes what SPE activities should be conducted during software development, and when and how to conduct them. Figure 1 depicts the SPE process; the following paragraphs explain the figure.

First, performance engineers define the specific SPE assessments for the current life cycle phase. Assessments determine whether planned software meets its *performance requirements,* such as acceptable response times, throughput thresholds, or constraints on resource requirements. A specific, quantitative requirement is vital if analysts are to determine concretely whether that requirement can be met. A crisp definition of the performance requirements lets developers determine the most appropriate means of achieving requirements, and avoid spending time overachieving them.

Business systems specify performance requirements in terms of *responsiveness* as seen by the system users. Reactive systems specify timing requirements for event responses or system throughput requirements. Batch systems specify the batch

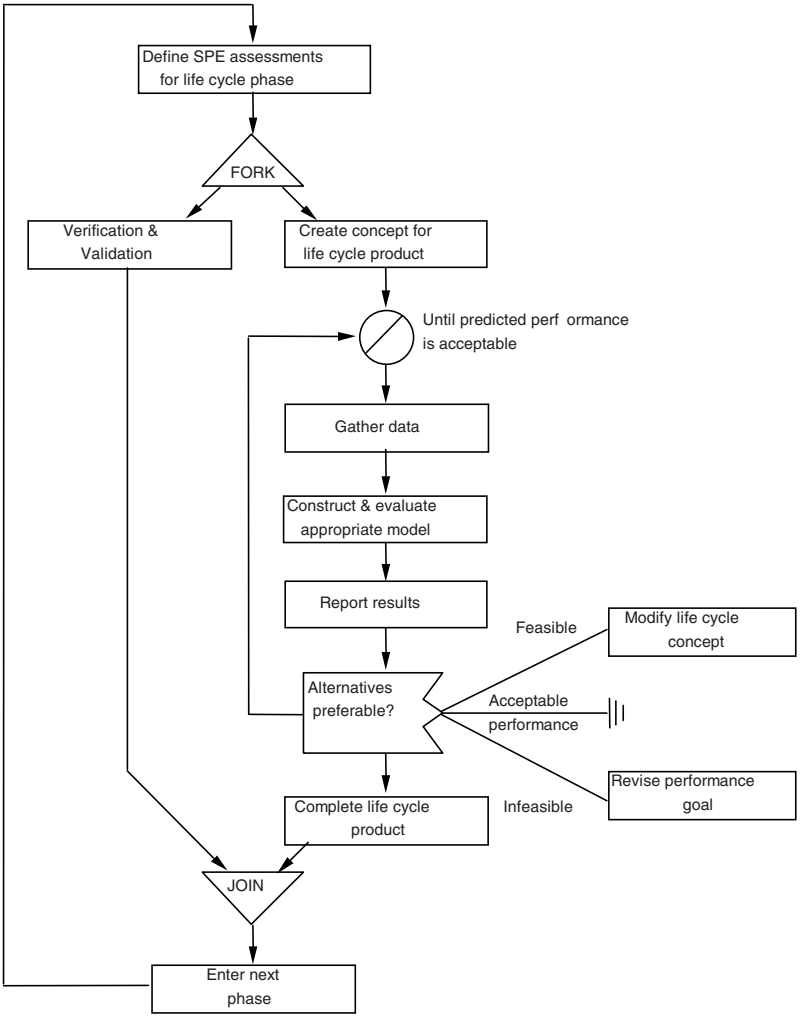


Fig. 1. Software Performance Engineering Process

window and the processing steps that must complete within it. Both the response time for a task and the number of work units processed in a time interval (throughput) are measures of responsiveness. Responsiveness does not necessarily imply efficient computer resource usage. Efficiency is addressed only if critical computer resource constraints must be satisfied.

After defining the requirements, designers create the *concept* for the life cycle product. For early phases the concept is the architecture – the software requirements and the high-level plans for satisfying requirements. In subsequent phases the concept is the design, the algorithms, the code, etc. Developers apply SPEs general principles, patterns and antipatterns (described later) for creating responsive architectures and designs.

Once the life cycle concept is formulated, analysts gather data sufficient for estimating the performance of the proposed concept. First, analysts need the projected *performance scenarios*: how the system will *typically* be used and the software components that will execute for those scenarios. In addition to the typical-usage analysis, reactive systems also examine worst-case and failure scenarios. Each performance scenario specifies the processing steps that execute when the scenario executes. The level of detail depends on the life cycle stage of the evaluation. Estimates of the resource usage of the processing steps complete the specifications. Section 3.1 provides more details on data requirements and techniques for gathering specifications.

Because the precision of the model results depends on the precision of the resource estimates, and because these are difficult to estimate very early in software development SPE calls for a *best and worst-case analysis strategy*. When there is high uncertainty about resource requirements, analysts use estimates of the lower and upper bound. Using them, the analysis produces an estimate of both the best and worst-case performance. If the best-case performance is unsatisfactory, they seek feasible alternatives. If the worst-case performance is satisfactory, they proceed with development. If the results are between the two extremes, models identify critical components whose resource estimates have the greatest effect, and focus turns to obtaining more precise data for them. A variety of techniques provide more precision, such as further refining the design concept and constructing more detailed models, or constructing performance benchmarks and measuring resource requirements for key elements.

An overview of the construction and evaluation of the performance models follows in Section 3.2. If the model results indicate that the performance is likely to be satisfactory, developers proceed. If not, analysts report quantitative results on the predicted performance of the original concept. If alternative strategies would improve performance, reports show the alternatives and their expected (quantitative) improvements. Developers review the findings to determine the cost-effectiveness of the alternatives. If a feasible and cost-effective alternative exists, developers modify the *concept* before the life cycle product is complete. If none is feasible as, for example, when the modifications would cause an unacceptable delivery delay, developers explicitly revise the performance requirement to reflect the expected degraded performance.

Vital and on-going activities of the SPE process are to *verify* that the models represent the software execution behavior, and *validate* model predictions against performance measurements. Reports compare the model specifications for the workload, the software components that execute, and resource requirements to actual usage and software characteristics. If necessary, analysts calibrate the model to represent the system behavior. They also examine discrepancies to update the performance predictions, and to identify the reasons for differences – to prevent similar problems occurring in the future. They produce reports comparing system execution model results (response times, throughput, device utilization, etc.) to measurements. Analysts study discrepancies, identify error sources, and calibrate the model as necessary. Model verification and validation should begin early and continue throughout the life cycle. In early stages, focus is on key performance factors; prototypes or benchmarks provide more precise specifications and

measurements as needed. As the software evolves, measurements serve to verify and validate the models.

This discussion outlined the steps for one design-evaluation "pass." The steps repeat throughout the life cycle. For each pass the evaluations change somewhat.

2.5 Guidelines for Creating Responsive Systems

Program efficiency techniques evolved first. Authors addressed program efficiency from three perspectives: efficient algorithms and data structures [60, 61]; efficient coding techniques [25, 56, 65]; and techniques for tuning existing programs to improve efficiency [15, 35, 36, 59].

Later, the techniques evolved to address large-scale *systems* of programs in early life-cycle stages when developers seek an architecture that will lead to a *system* with acceptable responsiveness [102, 103, 105]. During early stages, it is seldom the efficiency of machine resource usage that matters; it is the system *responsiveness*. Another distinction between system design and program tuning approaches is that program tuning transforms an inefficient program into a new "equivalent" program that performs the same function more efficiently. In system design, developers can transform *what* the software is to do as well as how it is to be done.

Smith [105, 122] defines nine principles: Performance Objective Principle, Instrumenting Principle, Centering Principle, Fixing Point Principle, Locality Principle, Processing Versus Frequency Principle, Shared Resources Principle, Parallel Processing Principle, and Spread-the-Load Principle. A quantitative analysis of the performance results of three of them is in Smith [102].

Both performance patterns and performance antipatterns have been proposed [119, 122]. The performance patterns present best practices for producing responsive software. They are: First Things First, Coupling, Batching, Alternate Routes, Flex-Time, and Slender Cyclic Functions. The performance antipatterns document common performance mistakes and provide solutions for them. They are: "god" Class, Excessive Dynamic Allocation, Circuitous Treasure Hunt, One-Lane Bridge, Traffic Jam, Unbalanced Processing, Unnecessary Processing, The Ramp, More is Less, Tower of Babel, Empty Semi's, and Domino Effect. The performance patterns and antipatterns complement and extend the performance principles. Each performance pattern is a realization of one or more of the performance principles while an antipattern violates one or more of them.

Lampson [64] also presented an excellent collection of hints for computer system design that addresses effectiveness, efficiency, and correctness. His efficiency hints are the type of folklore that has until recently been informally shared. Kopetz [62] presented principles for constructing real-time process control systems; some address responsiveness – all address performance in the more general sense.

Alter [3] and Kant [57] took a different approach; they used program optimization techniques to generate efficient programs from logical specifications. Search techniques identified the best strategy from various alternatives for choices such as data set organizations, access methods, and computation aggregations.

The principles, patterns, and antipatterns *supplement* performance assessment rather than replace it. Performance improvement has many tradeoffs – a local performance improvement may adversely affect overall performance. The quantitative

methods covered in section 3 provide the data required to evaluate the net performance effect to be weighed against other aspects of correctness, feasibility, and preferability.

3 Quantitative Methods for SPE

The quantitative methods prescribe the data required to conduct the performance assessment; techniques for gathering the data; the performance models; techniques for adapting models to the system evolution; and techniques for verification and validation of the models.

3.1 Data Requirements

To create a software execution model analysts need: performance requirements, workload definitions, software execution characteristics, execution environment descriptions, and resource usage estimates. An overview of each follows.

Precise, quantitative metrics, are vital to determine concretely whether or not *performance requirements* can be met. For business applications, both interactive performance requirements and batch window requirements must be met. For interactive transactions, the requirements specify the response time or throughput required. Specifications define the external factors that impact attainment, such as the time of day, the number of concurrent users, whether the requirement is an absolute maximum, a 90^{th} percentile, and so on. For reactive systems, timing constraints specify the maximum time between an event and the response. Some reactive systems have throughput requirements for the number of events processed in a time interval. Some have bounds on the utilization of computer resources used, such as a maximum of 50% CPU utilization.

Workload definitions specify the *performance scenarios* of the new software. For user interactions, scenarios (initially) specify the interactions expected to be the most frequently used. Later in the life cycle, scenarios also cover resource intensive transactions. Interactive workload definitions identify the performance scenarios and specify their workload intensity: the request arrival rates, or the number of concurrent users and the time between their requests (think time). Batch workload definitions identify the programs on the critical path, their dependencies, and the data volume to be processed. In addition to performance scenarios of typical uses, reactive systems represent scenarios of time-critical functions, and worst-case operating conditions.

Performance scenarios are usually easy to derive because many new systems replace either a manual process or a previous implementation of an automated system. It may be difficult to identify performance scenarios for revolutionary new functions or for Internet applications where the number of potential requests is unlimited and highly variable. Recent work by Gunther uses statistical techniques for forecasting demand for Internet applications [45].

Software processing steps identify components of the software system to be executed for each performance scenario. The software processing steps identify: software components most likely to be invoked when users request the corresponding performance scenario; the number of times they are executed or their probability of

execution; and their execution characteristics, such as the database tables used, and screens read or written. Reactive systems initially specify the likely execution paths, as well as less likely execution paths such as the worst-case path.

Software processing steps are relatively easy to identify when developed as the software evolves. It is more difficult when models are initially constructed after the software has been built, particularly for object-oriented systems. Hrishuk and Rolia developed techniques for constructing performance models from measurements of existing object-oriented systems [51, 52].

The *execution environment* defines the computer system configuration, such as the CPU, the I/O subsystem, the network elements, the operating system, the database management system, and middleware components. It provides key computer system and network devices for the underlying queueing network model and defines resource requirements for frequently used service routines.

The execution environment is usually the easiest information to obtain. Performance modeling tools automatically provide much of it, and most capacity and performance analysts are familiar with the requirements.

Resource usage estimates determine the amount of service required of key devices in the computer system configuration. For each software-processing step in the performance scenario, analysts need: the approximate number of instructions executed (or CPU time required); the number of physical I/Os; and use of other key devices such as the network (number of messages and the amount of data), etc. For database applications, the database management system (DBMS) accounts for most of the resource usage, so the number of database calls and their characteristics are necessary. Early life cycle requirements are tentative, difficult to specify, and likely to change, so SPE uses upper and lower bound estimates to identify problem areas or software components that warrant further study to obtain more precise specifications. Later, the models study the performance sensitivity to various parameter values.

There are four sources of values for resource usage data:

- measurements of parts of the software that are already implemented, such as basic services, existing

components, a design prototype or an earlier version [130],

- compiler output from existing code,
- demand estimates (CPU, I/O, etc.) based on designer and analyst judgment and reviews,
- "budget" figures, estimated from experience and the performance requirements, may be used as demand *objectives* for designers to meet (rather than as estimates for the code they will produce).

The sources at the top of the list are only useful for parts of the system that are actually running, so early analysis implies at least some of the numbers will be estimated.

It is seldom possible to get precise information for all these specifications early in the software's life cycle. Rather than waiting to model the system until it is available (i.e., in implementation or testing), SPE advocates gathering guesses, approximations, and bounded estimates to begin, then augmenting the models as information becomes available.

Because one person seldom knows all the information required for the software performance models, *performance walkthroughs* provide most of the life cycle data [105] [122]. Performance walkthroughs are closely modeled after design and code walkthroughs. In addition to software specialists who contribute software plans they bring together users who contribute workload, use case, and scenario information, and technical specialists who contribute computer configuration and resource requirements of key subsystems such as DBMS and communication paths. The primary purpose of a performance walkthrough is data gathering rather than a critical review of design and implementation strategy.

3.2 Performance Models

Two SPE models provide the quantitative data for SPE: the *software execution model* and the *system execution model*. The software execution model represents key facets of software execution behavior. The model solution quantifies the computer resource requirements for each performance scenario. The system execution model represents computer system resources with a network of queues and servers. The model combines the performance scenarios and quantifies overall resource utilization and consequent response times of each scenario.

There are several forms of software models for SPE. The following are the most common:

- The execution graphs of [105, 122] represent the sequence of operations, including precedence, looping, choices, synchronization, and parallel execution of flows. This is representative of a large family of models such as SP [131], task graphs (used in embedded and hard-real-time systems), activity diagrams, etc. Automated tools such as *SPE•ED* [63] and HIT [9] capture the data and reduce it to formal workload parameters.
- Communicating state machines are captured in software design languages such as UML, SDL and ROOM, and in many CASE tools, to capture sequence. Some efforts have been made to capture performance data in this way [32, 138] by capturing scenarios out of the operation of the state machines, or [128] by annotating the state machine and simulating the behavior directly.
- Petri nets and stochastic process algebras are closely related to state machines and also represent behavior through global states and transitions. Solutions are by Markov chain analysis in the state space, or by simulation.
- Annotated code represents the behavior by a code skeleton, with abstract elements to represent decisions and operations, and performance annotations to convey parameters. The code may be executed on a real environment, in which case we may call it a performance prototype, or by a simulator (a performance simulation) [5], or it may be reduced to formal workload parameters [73].

Component-based system assembly models, with components whose internal behavior is well understood, has been used to create simulation models automatically in Hyperformix models [54], and to create analytic layered queueing models [87, 139].

Execution graph models are one type of software execution model. A graph represents each performance scenario. Nodes represent processing steps of the

software; arcs represent control flow. The graphs are hierarchical with the lowest level containing complete information on estimated resource requirements.

A static analysis of the graphs yields mean, best- and worst-case response times of the scenarios. The static analysis makes the optimistic assumption that there are no other jobs on the host configuration competing for resources. Simple, graph-analysis algorithms provide the static analysis results [10, 19, 20, 105]. The static analysis is compared to the performance requirement and any problems are resolved before proceeding to the system execution model.

An example of a *SPE•ED* software execution model is in Figure 2 [63]. It shows a simple automated teller machine interaction. The software execution model is hierarchical, the lower levels in the model are shown in the small navigation boxes at the right of the screen. Colors on the screen show the connection between a node in a software model and its details in the navigation box. The figure also shows the end-to-end response time for one ATM user, and colors show the relative time spent in each step. Analysts use results such as these to identify problem areas, then explore alternatives to correct problems by modifying the model and comparing solutions.

Next, the system execution model solution yields the following additional information:

- the effect of resource contention on response times, throughput, device utilizations and device queue lengths.
- sensitivity of performance metrics to variations in workload composition
- effect of new software on service level objectives of other existing systems
- identification of bottleneck resources
- comparative data on performance improvements to the workload demands, software changes, hardware upgrades, and various combinations of each.

To construct and evaluate the system execution model, analysts use performance-modeling tools that represent the key computer resources with a network of queues. Environment specifications provide device parameters (such as CPU size and processing speed). Workload parameters and service requests come from the resource requirements computed from the software execution models. Analysts solve the model, check for reasonable results, and then examine the model results. If the results show that the system fails to meet performance requirements, analysts identify bottleneck devices and identify software components that have high usage of those devices. After identifying alternatives to the software plans or the computer configuration, analysts evaluate the alternatives by making appropriate changes to the software or system model and repeating the analysis steps.

Queueing network models are relatively lightweight and give basic analytic models, which solve quickly. They have been widely used [73, 115]. However the basic forms of queueing network model are limited to systems that use one resource at a time. Extended queueing models, that describe multiple resource use, are more complex to build and take longer to solve. *Layered queueing* is a framework for extended queueing models that can be built relatively easily, and which incorporates many forms of extended queueing systems [87, 139].

Bottleneck or bounds analysis of queueing networks is even faster than a full solution and has been widely used. However, when there are many classes of users or

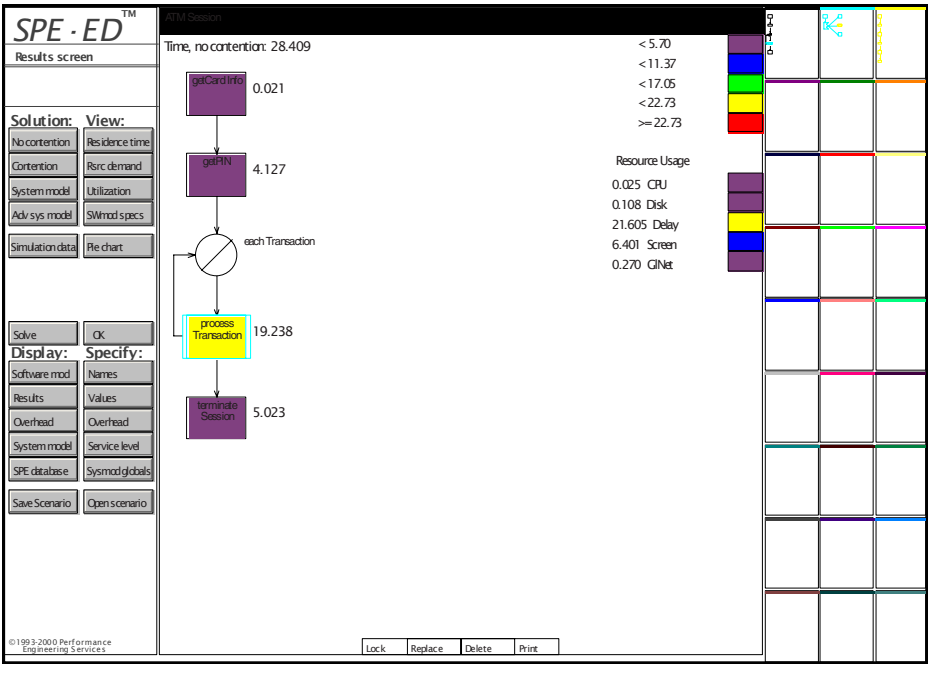


Fig. 2. Software Performance Model

of workloads (which is the case in most complex systems) the bounds are difficult to calculate and interpret. The difficulty with queueing models is expressiveness; they limit the behavior of the system to tasks providing service to requests in a queue. They cannot express the logic of intertask protocols, for instance.

Petri nets and other state-based models can capture logically convoluted protocol interactions that cannot be expressed at all in queueing models. They can in principle express any behavior whatever. Further, some of them, such as stochastic process algebra, include mechanisms for combining subsystems together to build systems. Their defect is that they require the storage of probabilities over a state space that may easily have millions of states, growing combinatorially with system size. They do not, and cannot scale up. However the size of system that can be solved has grown from thousands of states to millions, and interesting small designs can be analyzed this way.

Embedded and hard-real-time systems often use a completely different kind of model, a schedulability model, to verify if a schedule can be found that meets the deadlines. These are considered briefly below.

Simulation is the always-available fallback from any formulation. Some kinds of models are designed specifically for simulation, such as the performance prototype from code. Simulation models are heavyweight in the execution time needed to produce accurate results, and can give meaningless results in the hands of non-experts who do not know how to assess their precision or their accuracy. When models can be solved by both analytic queueing techniques and by simulation, the simulation time

may easily be three orders of magnitude greater. However simulation can always express anything that can be coded, so it never limits the modeler in that way.

System execution models based on the network of queues and servers can be solved with analytic techniques in a few seconds. Thus analysts can conduct many tradeoff studies in a short time. Analytic solution techniques generally yield utilizations within 10 percent, and response times within 30 percent of actual. Thus, they are well suited to early life cycle studies when the primary objective is to identify feasible alternatives and rule out alternatives that are unlikely to meet performance requirements. The resource usage estimates that lead to the model parameters are seldom sufficiently precise to warrant the additional time and effort required to produce more realistic models. Even reactive systems benefit from this intermediate step – it rules out serious problems before proceeding to more realistic models.

Early studies use simple SPE models. The SPE goal is to find problems with the simplest possible model. Experience shows that simple models can detect serious performance problems very early in the development process. The simple models isolate the problems and focus attention on their resolution. They are useful to evaluate many early architecture and design alternatives because they are easily formulated and quickly solved. After they serve this primary purpose, analysts augment them as the software evolves to make more realistic performance predictions. Advanced system execution models are usually appropriate when the software reaches the detail-design life cycle stage. Even when it is easy to incorporate the additional execution characteristics earlier, it is better to defer them to the advanced system execution model. It is seldom easy, however, and the time to construct, solve, and evaluate the advanced models usually does not match the input data precision early in the life cycle.

Facets of execution behavior such as usage of passive resources, complex execution environments, or tightly coupled models of software and system execution are represented in the advanced system execution model. It augments the elementary system execution model with additional types of constructs. Then procedures specify how to calculate corresponding model parameters from software models, and how to solve the advanced models. SPE methods specify "checkpoint evaluations" to identify those aspects of the execution behavior that require closer examination [105]. Simulation methods usually provide solutions for the advanced system models.

Details of these models are beyond the scope of this chapter. Introduction to system and advanced system execution models are available in books [44, 55, 66, 72] as well as other publications. The *Proceedings of the ACM SIGMETRICS Conferences* and the *Performance Evaluation Journal* report recent research results in advanced systems execution models. The *Proceedings of the Performance Petri Net Conferences* also report relevant results.

3.3 Verification and Validation

Another vital part of SPE is continual verification of the model specifications and validation of model predictions (V&V). Model verification and validation are ongoing activities that proceed in parallel with the construction and evaluation of the models. Model *verification* is aimed at determining whether the model predictions are an accurate reflection of the software's performance. It answers the question, "Are we

building the model right?" For example, are the estimates of resource requirements reasonable? Model *validation* is concerned with determining whether the model accurately reflects the execution characteristics of the software. It answers the question, "Are we building the right model?" The model should faithfully represent the evolving system. For SPE, it is particularly important to detect any model omissions as soon as possible.

Early V&V is needed when model results suggest that major changes are needed. The V&V effort matches the impact of the results and the importance of performance to the project. When performance is critical, or major software changes are indicated, analysts identify the critical components, implement or prototype them, and measure. Measurements verify resource usage and path execution specifications and validate model results. Early V&V is important even when predicted performance is good. Performance analysts drive the resource usage specifications, and analysts tend to be optimistic about how functions will be implemented, and about their resource requirements. Resource usage of the actual system often differs significantly from the optimistic specifications.

Performance engineers interview users, designers, and programmers to confirm that usage will be as expected, and that designed and coded algorithms agree with model assumptions. They adjust models when appropriate, revise predictions, and give regular status reports. They also perform sensitivity analyses of model parameters and determine thresholds that yield acceptable performance. Then, as the software evolves and code is produced, they measure the resource usage and path executions and compare them with these thresholds to get early warning of potential problems. As software increments are deployed, measurements of the workload characteristics yield comparisons of specified scenario usage to actual and show inaccuracies or omissions. Analysts calibrate models and evaluate the effect of model changes on earlier results. As the software evolves, measurements replace resource estimates in the SPE models.

V&V is crucial to SPE. It requires the comparison of multiple sets of parameters for heavy and light loads to corresponding measurements. The model precision depends on how closely the model represents the key performance drivers. It takes vigilance to make sure they match.

4 Example

To illustrate the process of modeling and evaluating the performance of a software design, we will use an example based on an automated teller machine (ATM). This example is based on a real-world development project; it has been simplified for this presentation, and some details have been changed to preserve anonymity.

> The ATM accepts a bank card and requests a personal identification number (PIN) for user authentication. Customers can perform any of three transactions at the ATM: deposit cash to an account, withdraw cash from an account, or request the available balance in an account. A customer may perform several transactions during a single ATM session. The ATM communicates with a computer at the host bank, which verifies the customer-account combination and processes the

transaction. When the customer is finished using the ATM, a receipt is printed for all transactions, and the customer's card is returned.

The following sections illustrate the application of the SPE process to the ATM.

4.1 Assess Performance Risk

The performance risk in constructing the ATM itself is small. Only one customer uses the machine at a time, and the available hardware is more than adequate for the task. Consequently, the amount of SPE effort on this project will be small. However, the host software (considered later) must deal with a number of concurrent ATM users, and response time there is important, so a more substantial SPE effort is justified.

4.2 Identify Critical Use Cases

We begin with the use case diagram for the ATM shown in Figure 3. As the diagram indicates, several use cases have been identified: Operator-Transaction (e.g., reloading a currency cassette), CustomerTransaction (e.g., a withdrawal), and CommandFunctions (e.g., to go off-line). Clearly, CustomerTransaction is the critical use case, the one that will most affect the customer's perception of the ATM's performance.

4.3 Select Key Performance Scenarios

We therefore select the CustomerTransaction as the first performance scenario to consider. This scenario represents typical, error-free customer transactions from the CustomerTransaction use case. Later, after we confirm that the architecture and design are appropriate for this scenario, we will consider additional scenarios. To evaluate the scenario, we need a specification for the *workload intensity*—that is, the number of CustomerTransactions or their arrival rate during the peak period.

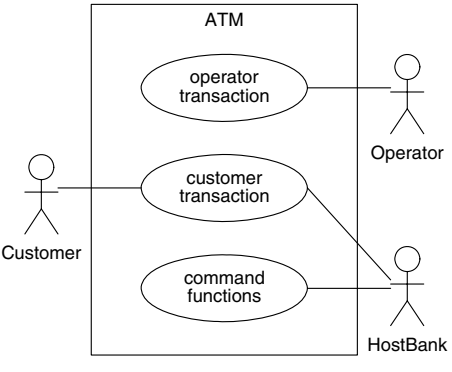


Fig. 3. ATM Use Cases

Figure 4 shows a scenario for customer transactions on the ATM. The notation used is a UML 2 sequence diagram. This scenario indicates that, after inserting a card and entering a PIN, a customer may repeatedly select transactions which may be deposits, withdrawals, or balance inquiries. The rounded rectangles indicate that the details of these transactions are elaborated in additional sequence diagrams.

The scenario in Figure 4 combines the customer transactions of deposit, withdrawal, and balance inquiry. We combine them because we want to model what a customer does during an ATM session, and a customer may request more than one transaction during a single session. While we don't know exactly which transaction(s)

a user will request, we can assign probabilities to each type of transaction based on reasonable guesses or actual measurements of customer activities.

We could also represent the information in Figure 4 using a UML activity diagram. However, we have found that the sequence diagram notation is more familiar to software developers, and is easier to translate to a software execution model**.**

4.4 Establish Performance Requirements

As a bank customer, what response time do you expect from an ATM? Historically, performance requirements have been based on "time in the (black) box," that is, the time from the arrival of the (complete) request to the time the response leaves the host computer. That approach was used to separate things outside the control of the software (e.g., the time for the user to enter information, network congestion, and so on) from those that are more directly influenced by the software itself. If we take this approach for the ATM, a reasonable performance requirement would be one second for the portion of the time on the host bank for each of the steps processDeposit, processWithdrawal, and processBalanceInquiry.

However, for SPE we prefer to expand the scope to cover the end-to-end time for a customer to complete a business task (e.g., an ATM session). Then, the results of the analysis will show opportunities to accomplish business tasks more quickly by reducing the number and type of interactions with the system, in addition to reducing the processing "in the box." A reasonable performance requirement for this scenario might be 30 seconds or less to complete the (end-to-end) ATM session.

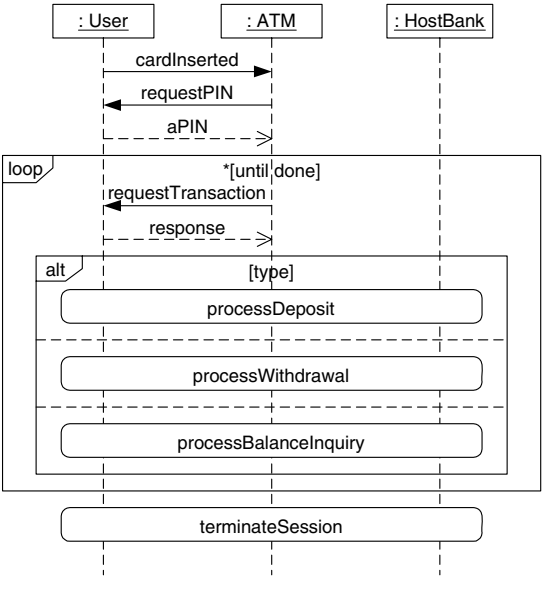


Fig. 4. Customer Transaction Scenario

4.5 Construct Performance Models

The models for evaluating the performance of the ATM are based on the key scenarios identified earlier in the process. These *performance scenarios* represent the same processing as the sequence diagrams using execution graphs.

Figure 5 shows the execution graph that corresponds to the ATM scenario in Figure 4. The rectangles indicate processing steps; those with bars indicate that the processing step is expanded in a subgraph. Figure 6 shows the expansion of the

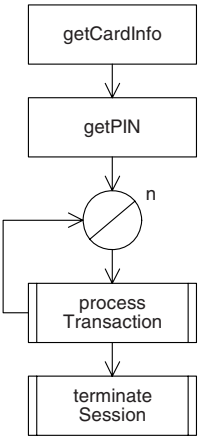


Fig. 5. ATM Execution Graph

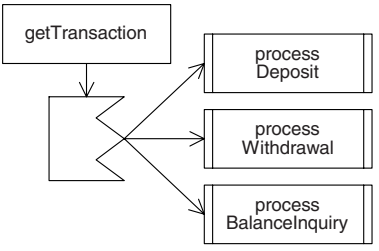


Fig. 6. Expansion of processTransaction

processTransaction step; expansion of the other steps is not shown here. The circular node indicates repetition, while the jagged node indicates choice.

The execution graph in Figure 5 expresses the same scenario as the sequence diagram in Figure 4 . After inserting a card (to provide customer information) and entering a PIN, a customer may repeatedly select transactions, which may be deposits, withdrawals, or balance inquiries. Here, the number of transactions that a customer may perform is indicated by the parameter *n*.

4.6 Determine Software Resource Requirements

The types of software resources will differ depending on the type of application and the operating environment. The types of software resources that are important for the ATM are:

- Screens—the number of screens displayed to the ATM customer
- Host—the number of interactions with the host bank
- Log—the number of log entries on the ATM machine
- Delay—the relative delay in time for other ATM device processing, such as the cash dispenser or receipt printer

Software resource requirements are application-technology specific. Different applications specify requirements for different types of resources. For example, a system with a significant database component might specify a software resource called “DBAccesses” and specify the requirements in terms of the number of accesses.

We specify requirements for each of these resources for each processing step in the execution graph, as well as the probability of each case alternative and the number of loop repetitions. Figure 7 shows the software resource requirements for processWithdrawal.

4.7 Add Computer Resource Requirements

We must also specify the *computer resource requirements* for each software resource request. The values specified for computer resource requirements connect the values for software resource requirements to device usage in the target environment. The computer resource requirements also specify characteristics of the operating environment, such as the types of processors/devices, how many of each, their speed, and so on.

Table 1 contains the computer resource requirements for the ATM example. The names of the devices in the ATM unit are in the first row, while the second row specifies how many devices of each type are in the facility. The third row is a comment that describes the unit of measure for the values specified for the software processing steps. The next four rows are the names of the software resources specified for each processing step, and the last row specifies the service time for the devices in the computer facility.

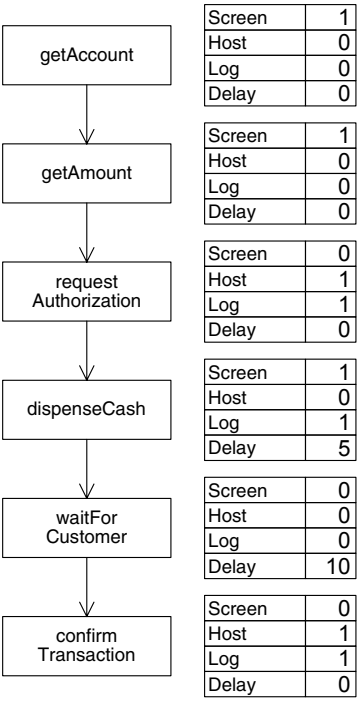


Fig. 7. Software Resource Requirements for processWithdrawal

The values in the center section of the table define the connection between software resource requests and computer device usage. The Display "device" represents the time to display a screen and for the customer to respond to the prompt.

Table 1. Overhead Matrix

Devices	CPU	Disk	Display	Delay	Net
Quantity	1	1	1	1	1
Service Units	Sec.	Phys.I/O	Screens	Units	Msgs.
Screen	0.001		1		
Host	0.005			3	2
Log	0.001	1			
Delay				1	
Service Time	1	0.02	1	1	0.05

The 1 in the Display column for the Screen row means that each screen specified in the software model causes one visit to the Display delay server. We arbitrarily assume this delay to be one second (in the service time row). Similarly, each Host and Delay specification in the software model results in a delay before processing the next step. We assume the Host delay is 3 seconds; other delays are specified in 1-second increments. Each Host request also sends 1 message via the Net and receives 1 reply message. Each message takes an average of 0.05 second. These values may be measured, or estimates could be obtained by constructing and evaluating more detailed models of the host processing required.

Thus, each value specified for a processing step in the software model generates a demand for service from one or more devices in a facility. The computer resource requirements define the devices used and the amount of service needed from each device. The demand is the product of the software model value times the value in the overhead matrix cell times the service time for the column.

4.8 Evaluate the Models

We begin by solving the software model. This solution provides a "no contention" result. Here, we find that the total end-to-end time for the scenario is approximately 28 seconds, and most of that is due to the delays at the ATM unit for customer interactions and processing (in Fig. 2). This is a best-case result and, in this case, confirms that a single ATM will complete in the desired time. Because it is close to exceeding the requirement, however, we may want to examine alternatives to reduce the end-to-end time. We should also examine the sensitivity of the results to our estimates. In addition, further studies will examine the host bank performance when there are multiple ATMs whose transactions could produce contention for computer resources. This will affect the time to handle Host requests.

4.9 Verify and Validate the Models

We need to confirm that the performance scenarios that we selected to model are critical to performance, and confirm the correctness of the workload intensity specifications, the software resource specifications, the computer resource specifications, and all other values that are input into the model. We also need to make sure that there are no large processing requirements that are omitted from the model. To do this, we conduct measurement experiments on the operating environment, prototypes, and analogous or legacy systems early in the modeling process. We measure evolving code as soon as viable. SPE suggests using early models to identify components critical to performance, and implementing them first. Measuring them and updating the model estimates with measured values increases precision in key areas early.

5 Status and Future of SPE

Since computers were invented, the attitude persists that the next hardware generation will offer significant cost-performance enhancements, so it will no longer be necessary to worry about performance. There was a time, in the early 1970s, when

computing power exceeded demand in most environments. The cost of achieving performance requirements, with the tools and methods of that era, made SPE uneconomical for many batch systems – its cost exceeded its savings. Today's methods and tools make SPE the appropriate choice for new systems, especially those with high visibility such as Web applications.

Will tomorrow's hardware solve all performance problems and make SPE obsolete? It has not happened yet. Hardware advances merely make new software solutions feasible, so software size and sophistication offset hardware improvements. There is nothing wrong with using more powerful hardware to meet performance requirements, but SPE suggests evaluating all options early, and selecting the most effective one. Thus hardware may be the solution, but it should be explicitly chosen – early enough to enable orderly procurement. SPE still plays a role.

The following sections examine research progress on SPE tasks and on tools that facilitate them.

5.1 Performance Risk Assessment

There has been a significant amount of research in software risk assessment, but little of it specifically addresses performance risks. Performance risk assessment determines both the probability that a performance failure will occur and projects its severity, that is, the cost of that failure. This information can be used to make a business case for SPE. Initial work in this area is reported in [134]. It is difficult to collect this data because companies are reluctant to publish information about failures. More work is needed to establish a data bank of information about the cost of failures and their frequency.

5.2 Workload Selection

There was a significant amount of work in workload characterization for system execution models. Early work measured computer resource usage of existing software and applied a clustering technique to formulate homogeneous workloads [34, 47]. The challenge was to forecast future workloads for software in development. Today the UML has facilitated the identification of use cases and their scenarios. Interaction with users helps to identify the performance scenarios to be represented in the SPE models. There is little additional research needed in this area. Perhaps an expert system could assist developers and users in identifying the performance scenarios automatically. Perhaps analysis of measurements of the deployed system could help performance analysts determine if the original models need to be updated to reflect newer usage patterns.

5.3 Performance Requirements

Sometimes the performance requirements are clear, as in reactive systems where factors in the outer system determine deadlines for certain types of responses. In systems with human users they are rarely clear, because satisfaction is a moving target. As systems get better, users demand more of them, and hope for faster responses.

In most systems there are several types of response, with different requirements. Users can be asked to describe performance requirements, to identify the types and the acceptable delays and capacity limitations. An experienced analyst must review their responses with the users, because they may be unnecessarily ambitious (and thus potentially costly), or else not testable. This review may have to be repeated as the evaluation proceeds and the potential costs of achieving the goals becomes clearer.

Experience in defining performance requirements is not well documented, however it seems clear that they are often not obtained in any depth, or checked for realism, consistency and completeness, and that this is the source of many performance troubles with products. One problem is that users expect the system to modify how they work, so they do not really know precisely how they will use it. Another is that while developers have a good notion of what constitutes well-defined functional requirements many are naive about performance requirements.

Research is needed on questions such as tests for realism, testability, completeness and consistency of performance requirements; on methodology for capturing them, preferably in the context of a standard software notation such as UML, and on the construction of performance tests from the requirements.

5.4 Performance Models

There has been much research on various facets of SPE models [126]. Recently researchers have concentrated on automatic translation of UML models into a performance model. The establishment of a standard for annotating UML diagrams with performance specifications has facilitated this approach [76]. This work is described later.

Performance modeling is a fairly mature area. Recent advances in automatic translation make it easier for developers to construct and evaluate their own models. Research is needed that will help them identify critical parts of the *software* and focus on simple models of those parts, particularly for complex distributed systems.

5.5 Resource Requirements

The software execution structure is relatively easy to translate into a model. Obtaining specifications for resource requirements such as for CPU time, disk operations, network services and so forth, is still a stumbling block. The UML standard is a start [76], but it gives no guidance to developers on how to obtain the values to specify.

There has been considerable experience with estimation using expert designer judgment [105, 122], and it works well provided it has the participation (and encouragement) of a performance expert on the team. Few designers are comfortable in setting down inexact estimates, and on their own they have a tendency to chase precision, for example by creating prototypes and measuring them. Nonetheless the estimates are useful even if they turn out later to have large errors, since the performance prediction is usually sensitive to only a few of the values.

Parameter estimation is an important area, yet it is difficult to plan research that will overcome the difficulties. Systems based on re-using existing components offer an opportunity to use demand values found in advance from performance tests of their major functions, and stored in a demand database.

5.6 Model Solution Technology

In the early days of SPE evolution, a key issue was the need for fast model solutions. Much of the early model research was on approximate analytic solutions for complex systems. The speed of today's processors makes simulation solutions viable for a large class of models. Today, analytic solutions still play an important role in the analysis of software systems. There is no need to run a detailed simulation to get 99% confidence in the results when early design data estimates are imprecise. Serious flaws in architectures and designs can be detected and corrected with relatively simple models [133].

The advent of distributed systems, Web-based systems with heavy tailed distributions, and systems with QoS concerns such as loss and jitter challenge the modeling power of QNMs and their analytic solutions. While it is possible to solve these models with simulation methods, it is desirable to also use some quicker, simpler techniques for identifying and resolving problems.

There is a great deal of research in this area. Much of it is presented at the meetings of the Workshop on Software and Performance (WOSP). The challenge is to cast the results into a framework that is amenable to technology transfer to users who do not have extensive formal performance modeling skills. For example, less skilled users cannot effectively select from among "one of a kind" models customized to a narrow problem. They need a general approach that applies to a wide variety of problems.

5.7 Evaluate Results

If the model shows that the performance is unsatisfactory, analysts interpret the results to identify the source of the problems, then identify and evaluate alternatives. If the requirements are not met, the analysis will often point to changes that make it satisfactory. These may be changes to the execution platform, to the software design, or to the requirements.

Performance antipatterns characterize common software architecture and design problems and their solutions [121, 122, 124]. Changes to the software implementation may either reduce the cost of individual operations, or reduce the number of repetitions of an operation, as described in [122]

To diagnose system-level changes one must trace the causes of unsatisfactory delays and throughputs, back to delays and bottlenecks inside the system. Sometimes the causes are obvious system-related problems, for example a single bottleneck that delays all responses and throttles the throughput. Sometimes they are subtler, such as a network latency that blocks a process, making it busy for long unproductive periods and creating a bottleneck there. Examples are given in [31, 105, 122].

If the bottleneck is a processing device (CPU, network card, I/O channel, disk, or attached device of some kind) then the analysis can be modified to consider more powerful devices. If the cost of adapting the software or the environment is too high, one should finally consider the possibility that the requirements are unrealistic and should be relaxed.

5.8 Model Verification and Validation

Both verification and validation require measurement. In cases where performance is critical, it may be necessary to identify critical components, implement or prototype them early in the development process, and measure their performance characteristics. The model solutions help identify which components are critical.

There are some technical issues with validating the solutions to simulation models, such as confirming that the simulation run length is adequate. The primary concern for SPE models is model verification: whether the model parameters for non-existent software are sufficiently precise. This is not currently an active research area. If the modeling and analysis is to be conducted by developers rather than modeling gurus, however, we need to automate more of these tasks. Some of the associated measurement issues are discussed in the next section.

5.9 Facilitating SPE tasks

This category of research is aimed at making the task of conducting SPE easier. Several problems are described, and some preliminary work in these areas is reviewed.

5.9.1 Model Interchange Formats

A performance model interchange format (PMIF) is a common representation for system performance model data that can be used to move models among modeling tools [113, 118]. A user of several tools that support the format can create a model in one tool, and later move the model to other tools for further work without the need to laboriously translate from one tool's model representation to the other.

PMIF provides a common ground that all tools may use as an interface. Without it two tools would need to develop a custom import and export mechanism. A third tool would require a custom interface between each of those tools resulting in a $4 \cdot (N! / (2!(N-2)!))$ requirement for customized interfaces. With PMIF, tools export and import with the same format so only two customized interfaces (per tool) are required.

Work in this area began with the introduction of PMIF [113, 118] and a meta-model defining information requirements for SPE [132]. XML implementations were created for them in [109, 110]. A considerable amount of other work has addressed interchanges from UML to various types of performance models, such as [6, 30, 42, 43, 141]. Supporting tools were also developed, such as a PMIF semantic validator and Web Services to make the tools easier to use [39, 89].

This is an active and exciting area of research. One notable challenge in going from UML to S-PMIF is that the translated model is far more detailed than desirable. Performance models should abstract the essence of the processing details so that it is easier to analyze and evaluate the results. In general, many of the processing steps in an automatically generated model are not interesting from a performance standpoint, and the extra steps tend to "clutter" the model. This is a departure from the simple model strategy described earlier. In cases where the model is relatively small, as in early design stages, it may be easier to just create a new model and omit those details initially. Some techniques for "pruning" an automatically generated model would make it better suited for SPE.

5.9.2 Measurements for SPE

While there are numerous tools for measuring performance characteristics of computer systems, there are still some limitations in conducting measurement studies for SPE:

- Software is often developed on a different platform that it is deployed on, so measurements on the development platform are not indicative of the final performance, so problems may go undetected until late in development.
- It is sometimes difficult to measure the target platform. For example, high traffic systems may not tolerate the perturbation of measurement tools, and some reactive process control systems do not have measurement tools available.
- Functional test data is seldom representative of performance workloads. Either the volume of work is not representative, or the content used to test functionality does not reflect the key performance scenarios.
- Most tools are intended for system performance management, and it is difficult to get fine-grained data on software performance.
- It is seldom possible to get the SPE data from one tool, so the experiments to collect data require a great deal of work to coordinate the tests and the tools, then to interpret the data.

Since all modeling tools need similar data, it would be nice to agree on some standard specifications, e.g., data requirements and XML tags, and have measurement tools export the desired data in this format for automated use in various modeling tools. Model interchange formats are a good starting point because they specify the data needed for SPE. The specifications should address not only model creation, but also model validation.

5.9.3 System Specific Guidance

There has been work that provides guidance for developers of specific types of systems. It describes the particular problems in developing those systems, how to do the modeling and analysis, and advice for avoiding typical problems in those systems. Representative examples are in [31, 71, 93]. More work of this type would help new SPE practitioners.

5.9.4 Architecture Assessment

Many performance problems are introduced at the time the software architecture is selected. This problem has been addressed by defining a "packaged solution," known as Performance Assessment of Software Architectures (PASA^{SM}) [133]. It specifies the steps to be conducted for an architecture performance assessment so that it is more likely to be conducted and thus prevent serious problems from occurring.

5.9.5 Software Process Integration

The best SPE successes are achieved when the analysis steps are part of the software development process [53, 92, 123]. An informal description of how to integrate these steps with the Rational Unified Process (RUP) is in [122]. It still requires that the steps be customized to each development organization. There is usually a separation between the technical experts in performance engineering and the process specialists

in the software development organization. This hinders the close integration of SPE with development, and thus makes it less likely that the steps will be conducted.

5.9.6 Tools

Both research and development will produce the tools of the future. We seek better integration of the models and their analysis with software development and measurement tools. Ideally, developers should create and evaluate their own models (rather than interfacing with performance specialists to have them built). Then software changes would automatically update prediction models. Simple models can be transparent to designers – designers could click a button while formulating designs and view automatically generated predictions. Expert systems could automatically suggest alternatives. Visual user interfaces could make analysis and reporting more effective. Software measurement tools could automatically capture, reduce, interpret, and report data at a level of detail appropriate for designers. They could automatically generate performance tests then automate the verification and validation process by comparing specifications to measurements, and predictions to actual performance, and reporting discrepancies. Each of these tools could interface with an SPE database to store evolutionary design and model data and support queries against it.

While simple versions of each of these tools are feasible with today's technology, research must establish the framework for fully functional versions. Scenarios in object-oriented development tools are close to software execution models, and automatic translation for them is viable. A key problem with the translation approach is that development tools currently do not collect data, such as resource requirement specifications, that is essential for performance model solutions. Further research is needed to identify viable ways of providing this information, such as with expert systems, using analogous systems, etc. Other research questions are: How should performance models integrate with program generators – should one begin with models and generate code from them, or should one create the program and let underlying models select efficient implementations, or some other combination? How can expert systems detect problems? Can software automatically determine from software execution models where instrumentation probes should be inserted? Can software automatically reduce data to appropriate levels of detail? Can software automatically generate performance tests? Each of these topics represents extensive research projects.

5.10 SPE Methods

SPE methods should undergo significant change as its usage increases. The methods should be better integrated into the software development process, rather than an add-on activity. SPE should become better integrated into capacity planning as well. As they become integrated, many of the pragmatic techniques should be unnecessary (how to convince designers there is a serious problem, how to get data, etc.). The nature of SPE should then change. Performance walkthroughs will not be necessary for data gathering; they may only review performance during the course of regular design walkthroughs. The emphasis will change from finding and correcting design flaws to verification and validation that the system performs as expected. The process in Figure 1 currently specifies steps to be conducted by performance specialists. The

methods should evolve to empower developers to conduct their own SPE studies. Additional research into automatic techniques for measuring software designs is needed, for calibrating models, and for reporting discrepancies.

6 Summary and Conclusions

There has been a tremendous amount of research in the SPE field since it was first proposed as a discipline in 1981 [99]. This paper reviews the origins of SPE. It then summarizes the steps in the SPE process and the guidelines for creating responsive systems. It illustrates the SPE process with a case study. It then reviews recent research in the context of the steps in the SPE process, and other research that facilitates the SPE tasks.

This brief overview shows that there has been a great deal of progress, particularly in the modeling and analysis areas. There is still much more work to be done to make the tasks quicker and easier, to make SPE more accessible to software developers rather than requiring modeling gurus, and to make SPE more likely to be adopted and used in development organizations. Some of the problems could be resolved by making some changes in the way software performance is taught in universities.

Research challenges for the future are to extend the quantitative methods to model emerging hardware-software developments, to extend hardware-software measurement technology to support SPE, and to develop interdisciplinary techniques to address the more general definition of performance. The challenges for future technology transfer are to integrate SPE with software engineering process and tools, to shorten the time for SPE studies, to automate the sometimes-cumbersome SPE activities, and to evolve SPE to make it easy and economical for future environments.

References

[1] C.T. Alexander. *Performance Engineering: Various Techniques and Tools*. in *Proceedings Computer Measurement Group Conference*. 1986. Las Vegas, NV.

[2] W. Alexander and R. Brice. *Performance Modeling in the Design Process*. in *Proceedings National Computer Conference*. 1982. Houston, TX.

[3] S. Alter, *A Model for Automating File and Program Design in Business Application Systems.* Communications of the ACM, 1979. **22**(6): p. 345-353.

[4] G.E. Anderson, *The Coordinated Use of Five Performance Evaluation Methodologies.* Communications of the ACM, 1984. **27**(2): p. 119-125.

[5] R.L. Bagrodia and C. Shen, *MIDAS: Integrated Design and Simulation of Distributed Systems.* IEEE Transactions on Software Engineering, 1991. **17**(10): p. 1042-58.

[6] S. Balsamo and M. Marzolla. *Performance Evaluation of UML Software Architectures with Multiclass Queueing Network Models*. in *WOSP 2005*. 2005. Palma de Mallorca: ACM.

[7] F. Baskett, et al., *Open, Closed, and Mixed Networks of Queues with Different Classes of Customers.* Journal of the ACM, 1975. **22**(2): p. 248-260.

[8] H. Beilner, J. Mäter, and N. Weissenburg. *Towards a Performance Modeling Environment: News on HIT*. in *Proceedings 4th International Conference on Modeling Techniques and Tools for Computer Performance Evaluation*. 1988: Plenum Publishing.

[9] H. Beilner, J. Mäter, and C. Wysocki, *The Hierarchical Evaluation Tool HIT*, in *Performance Tools & Model Interchange Formats*, F. Bause and H. Beilner, Editors. 1995, Universität Dortmund, Fachbereich Informatik: D-44221 Dortmund, Germany. p. 6-9.

[10] B. Beizer, *Micro-Analysis of Computer System Performance*. 1978, New York, NY: Van Nostrand Reinhold.

[11] B. Beizer, *Software Performance*, in *Handbook of Software Engineering*, C.R. Vicksa and C.V. Ramamoorthy, Editors. 1984, Van Nostrand Reinhold: New York, NY. p. 413-436.

[12] T.E. Bell, ed. *Special Issue on Software Performance Engineering*. Computer Measurement Group Transactions. 1988.

[13] T.E. Bell, D.X. Bixler, and M.E. Dyer, *An Extendible Approach to Computer-aided Software Requirements Engineering*. IEEE Transactions on Software Engineering, 1977. **3**(1): p. 49-59.

[14] T.E. Bell and A.M. Falk. *Performance Engineering: Some Lessons From the Trenches*. in *Proceedings Computer Measurement Group Conference*. 1987. Orlando, FL.

[15] J.L. Bentley, *Writing Efficient Programs*. 1982, Englewood Cliffs, NJ: Prentice-Hall.

[16] S. Bernardi, S. Donatelli, and J. Merseguer. *From UML Sequence Diagrams and Statecharts to Analysable Petri Net Models*. in *Proc. Third Int. Workshop on Software and Performance (WOSP02)*. 2002. Rome: ACM Press.

[17] M. Bernardo, P. Ciancarini, and L. Donatelli. *AEMPA: A Process Algebraic Description Language for the Performance Analysis of Software Architectures*. in *Proc. Second Int. Workshop on Software and Performance (WOSP00)*. 2000. Ottawa: ACM Press.

[18] BMC, *BMC Software*, in *2101 City West Blvd.*, (713) 918-8800, www.bmc.com: Houston, TX 77042.

[19] T.L. Booth, *Performance Optimization of Software Systems Processing Information Sequences Modeled by Probabilistic Languages.* IEEE Transactions on Software Engineering, 1979. **5**(1): p. 31-44.

[20] T.L. Booth. *Use of Computation Structure Models to Measure Computation Performance*. in *Proceedings Conference on Simulation, Measurement, and Modeling of Computer Systems*. 1979. Boulder, CO.

[21] T.L. Booth, R.O. Hart, and B. Qin. *High Performance Software Design*. in *Proceedings Hawaii International Conference on System Sciences*. 1986. Honolulu, HI.

[22] T.L. Booth and C.A. Wiecek, *Performance Abstract Data Types as a Tool in Software Perfromance Analysis and Design.* IEEE Transactions on Software Engineering, 1980. **6**(2): p. 138-151.

[23] G. Brataas, *Performance Engineering Method for Workflow Systems: An Integrated View of Human and Computerised Work Processes*, in *Physics, Informatics, and Mathematics*. 1996, Norwegian University of Science and Technology: Trondheim, Norway. p. 237.

[24] R.J. Buhr, et al., *Software CAD: A Revolutionary Approach.* IEEE Transactions on Software Engineering, 1989. **15**(3): p. 234-249.

[25] D. Bulka and D. Mayhew, *Efficient C++: Performance Programming Techniques.* 2000: Addison Wesley Longman.

[26] J.P. Buzen, *Queueing Network Models of Multiprograming*. 1971, Harvard University.

[27] C.K. Chang, et al., *Modeling a Real-Time Multitasking System in a Timed PQ Net.* IEEE Transactions on Software Engineering, 1989. **6**(2): p. 46-51.

[28] CMG, *Computer Measurement Group*, in *PO Box 1124*, (800) 436-7264, www.cmg.org: Turnersville, NJ 08012.

[29] P. Crain and C. Hanson. *Web Application Tuning*. in *CMG*. 1999. Reno.
[30] A. D'Ambrogio. *A Model Transformation Framework for the Automated Building of Performance Models from UML Models*. in *Proc. 2005 Workshop on Software and Performance*. 2005. Palma de Mallorca: ACM Press.
[31] J. Dilly, et al. *Measurement Tools and Modeling Techniques for Evaluating Web Server Performance*. in *Proc. 9th Int. Conf. on Modelling Techniques and Tools*. 1997. St. Malo, France: Springer-Verlag.
[32] H. El-Sayed, D. Cameron, and C.M. Woodside. *Automated Performance Modeling from Scenarios and SDL Designs of Telecom Systems*. in *Proc. Int. Symposium on Software Engineering for Parallel and Distributed Systems (PDSE98)*. 1998. Kyoto, Japan.
[33] G. Estrin, et al., *SARA (System ARchitects' Apprentice): Modeling, Analysis, and Simulation Support for Design of Concurrent Systems.* IEEE Transactions on Software Engineering, 1986. **SE-12**(2): p. 293-311.
[34] D. Ferrari, *Workload Characterization and Selection in Computer Performance Measurement.* IEEE Computer, 1972. **5**(4): p. 18-24.
[35] D. Ferrari, *Computer Systems Performance Evaluation*. 1978, Englewood Cliffs, NJ: Prentice-Hall.
[36] D. Ferrari, G. Serazzi, and A. Zeigner, *Measurement and Tuning of Computer Systems*. 1983, Englewood Cliffs, NJ: Prentice-Hall.
[37] G. Fox. *Take Practical Performance Engineering Steps Early*. in *Proceedings Computer Measurement Group Conference*. 1987. Orlando, FL.
[38] G. Fox. *Performance Engineering as a Part of the Development Lifecycle for Large-Scale Software Systems*. in *Proceedings 11th International Conference on Software Engineering*. 1989. Pittsburgh, PA.
[39] D. García, et al. *Performance Model Interchange Format: Semantic Validation*. in *2006 International Conference on Software Engineering Advances*. 2006. Papeete Tahiti: INIRIA.
[40] H. Gomaa and D.A. Menasce, eds. *Session: From Specifications to Performance Models*. Second Int. Workshop on Software and Performance (WOSP00). 2000, ACM Press: Ottawa. 47-88.
[41] R.M. Graham, G.J. Clancy, and D.B. DeVaney, *A Software Design and Evalation System.* Communications of the ACM, 1973. **16**(2): p. 110-116.
[42] G. Gu and D. Petriu. *XSLT Transformation from UML Models to LQN Performance Models*. in *Proc. Workshop on Software and Performance*. 2002. Rome: ACM.
[43] G. Gu and D. Petriu. *From UML to LQN by XML Algebra-based Model Transformations*. in *WOSP 2005*. 2005. Palma de Mallorca: ACM.
[44] N. Gunther, *The Practical Performance Analyst: Performance-By-Design Techniques for Distributed Systems*. 1998: www.perfdynamics.com.
[45] N. Gunther. *E-Ticket Capacity Planning: Riding the E-Commerce Growth Curve*. in *Proc. Computer Measurement Group*. 2000. Orlando.
[46] C. Hanson, P. Crain, and S. Wigginton. *User and Computer Performance Optimization*. in *CMG*. 1999. Reno.
[47] G. Haring, *On State-dependent Workload Characterization by Software Resources.* ACM SIGMETRICS Performance Evaluation Review, 1982. **11**(4): p. 51-57.
[48] G. Hills, J.A. Rolia, and G. Serazzi. *Performance Engineering of Distributed Software Process Architectures*. in *Modelling Techniques and Tools for Computer Performance Evaluation*. 1995. Heidelberg, Germany: Springer.

[49] R.P. Hopkins, et al. *Two Approaches to Integrating UML and Performance Models*. in *Proc. Third Int. Workshop on Software and Performance (WOSP02)*. 2002. Rome: ACM Press.
[50] N.R. Howes. *Toward a Real-Time Ada Design Methodology*. in *Proceedings Tri-Ada 90*. 1990. Baltimore, MD.
[51] C. Hrischuk, J.A. Rolia, and C.M. Woodside. *Automatic Generation of a Software Performance Model Using an Object-Oriented Prototype*. in *Proceedings of the Third International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems*. 1995. Durham, NC.
[52] C.E. Hrischuk, et al., *Trace-Based Load Characterization for Generating Performance Software Models.* TSE, 1999. **25**(1): p. 122-135.
[53] P. Hughes. *Towards a Common Process Model for Systems Development and Performance Engineering*. in *Proc. Second Int. Workshop on Software and Performance (WOSP00)*. 2000. Ottawa, Canada: ACM Press.
[54] HyPerformix, *Inc.*, in *4301 West Bank Dr, Bldg A*, (512)328-5544, www.hyperformix.com: Austin, TX 78746.
[55] R. Jain, *Art of Computer Systems Performance Analysis*. 1991, New York, NY: John Wiley.
[56] P.J. Jalics, *Improving Performance The Easy Way.* Datamation, 1977. **23**(4): p. 135-148.
[57] E. Kant, *Efficiency in Program Synthesis*. 1981, Ann Arbor, MI: UMI Research Press.
[58] P. King and R. Pooley. *Derivation of Petri Net Performance Models from UML Specifications of Communications Software*. in *Modelling Tools and Techniques*. 2000. Schaumburg, IL: Springer.
[59] D.E. Knuth, *An Empirical Study of FORTRAN Programs.* Software Practice & Experience, 1971. **1**(2): p. 105-133.
[60] D.E. Knuth, *The Art of Computer Programming, Vol.1: Fundamental Algorithms, Third Edition*. 1997, Reading, MA: Addison-Wesley.
[61] D.E. Knuth, *The Art of Computer Programming, Vol.3: Sorting and Searching, Second Edition*. 1998, Reading, MA: Addison-Wesley.
[62] H. Kopetz. *Design Principles of Fault Tolerant Real-Time Systems*. in *Proceedings Hawaii International Conference on System Sciences*. 1986. Honolulu, HI.
[63] L&S, *Computer Technology, Inc., Performance Engineering Services Division*, in *# 110, PO Box 9802*, (505) 988-3811, www.spe-ed.com: Austin, TX 78766.
[64] B.W. Lampson, *Hints for Computer System Design.* IEEE Software, 1984. **2**(1): p. 11-28.
[65] C. Larman and R. Guthrie, *Java 2 Performance and Idiom Guide*. 2000, Upper Saddle River, NJ: Prentice Hall PTR.
[66] E.D. Lazowska, et al., *Quantitative System Performance: Computer System Analysis Using Queuing Network Models*. 1984, Englewod Cliffs, NJ: Prentice-Hall, Inc.
[67] E. LeMer. *MEDOC: A Methodology for Designing and Evaluating Large-Scale Real-Time Systems*. in *Proceedings National Computer Conference, 1982*. 1982. Houston, TX.
[68] S. Levi and A.K. Agrawala, *Real-Time System Design*. 1990, New York, NY: McGraw-Hill.
[69] K. Lor and D.M. Berry, *Automatic Synthesis of SARA Design Models from System Requirements.* IEEE Transactions on Software Engineering, 1991. **17**(12): p. 1229-1240.

[70] C.R. Martin, *An Integrated Software Performance Engineering Environment*. 1988, Duke University.
[71] D.A. Menascé and V.A.F. Almeida, *Scaling for E-Business: Technologies, Models, Performance, and Capacity Planning*. 2000: Prentice Hall.
[72] D.A. Menascé, V.A.F. Almeida, and L.W. Dowdy, *Capacity Planning and Performance Modeling*. 1994, Englewood Cliffs, NJ: PTR Prentice Hall. 412.
[73] D.A. Menascé and H. Gomaa. *On a Language Based Method for Software Performance Engineering of Client/Server Systems*. in *Workshop on Software and Performance*. 1998. Santa Fe, NM: ACM.
[74] K.M. Nichols, *Performance Tools.* IEEE Software, 1990. **7**(3): p. 21-30.
[75] K.M. Nichols and P. Oman, *Special Issue in High Performance.* IEEE Software, 1991. **8**(5).
[76] OMG, *UML Profile for Schedulability, Performance and Time, formal/03-09-01*. 2003, OMG Full Specification.
[77] A. Opdahl. *A CASE Tool for Performance Engineering During Software Design*. in *Proceedings Fifth Nordic Workshop on Programming Environmental Research*. 1992. Tampere, Finland.
[78] A. Opdahl and A. Sølvberg. *Conceptual Integration of Information System and Performance Modeling*. in *Proceedings Working Conference on Information System Concepts: Improving the Understanding*. 1992.
[79] M. Paterok, R. Heller, and H. deMeer. *Performance Evaluation of an SDL Run Time System - A Case Study*. in *Proceedings 5th International Conference on Modeling Techniques and Tools for Computer Performance Evaluation*. 1991. Torino, Italy.
[80] D. Petriu and C.M. Woodside. *Software Performance Models from System Scenarios in Use Case Maps*. in *Proceedings 12th Int. Conf. Modeling Tools and Techniques*. 2002. London.
[81] R. Pooley. *The Integrated Modeling Support Environment*. in *Proceedings 5th International Conference on Modeling Techniques and Tools for Computer Performance Evaluation*. 1991. Torino, Italy.
[82] QEST, *Quantitative Evaluation of Systems*. 2004++, www.qest.org.
[83] B. Qin, *A Model to Predict the Average Response Time of User Programs.* Performance Evaluation, 1989. **10**: p. 93-101.
[84] QSP, *Quantitative System Performance*, in *7516 34th Ave N.*, A.M.a.A.P. QSP, Editor: Seattle, WA 98117-4723.
[85] W.E. Riddle, et al., *Behavior Modeling During Software Design.* IEEE Transactions on Software Engineering, 1978. **4**.
[86] J.A. Rolia, *Predicting the Performance of Software Systems*. 1992, University of Toronto.
[87] J.A. Rolia and K.C. Sevcik, *The Method of Layers.* IEEE Trans. on Software Engineering, 1995. **21**(8): p. 689-700.
[88] J.A. Rolia and V. Vetland. *Correlating Resource Demand Information with ARM Data for Application Services*. in *Int. Workshop on Software and Performance*. 1998. Santa Fe, NM: ACM.
[89] J. Rosselló, et al. *A Web Service for Solving Queueing Network Models Using PMIF*. in *Proc. Workshop on Software and Performance*. 2005. Palma de Mallorca: ACM Press.
[90] R.A. Sahner and K.S. Trivedi, *Performance and Reliability Analysis Using Directed Acyclic Graphs.* IEEE Transactions on Software Engineering, 1987. **13**(10): p. 1105-1114.

[91] J.W. Sanguinetti. *A Formal Technique for Analyzing the Performance of Complex Systems*. in *Proceedings Performance Evaluation Users Group 14*. 1978. Boston, MA.
[92] A. Schmietendorf, A. Scholz, and C. Rautenstrauch. *Evaluating the Performance Engineering Process*. in *Proc. Workshop on Software and Performance (WOSP2000)*. 2000. Ottawa, Canada.
[93] T. Schneider, *Performance Management of SAP Solutions*, in *Performance Engineering: State of the Art and Current Trends*, Dumke, et al., Editors. 2001, Springer Verlag LNCS 2047.
[94] K.C. Sevcik, C.U. Smith, and L.G. Williams, *Performance Prediction Techniques Applied to Electronic Commerce Systems*, in *Electronic Commerce Technology Trends: Challenges and Opportunities*, W. Kou and Y. Yesha, Editors. 2000, IBM Press.
[95] F. Sheikh and C.M. Woodside, *Layered Analytic Performance Modelling of a Distributed Database System.* ICDCS, 1997.
[96] H. Sholl and S. Kim. *An Approach to Performance Modeling as an Aid in Structuring Real-time, Distributed System Software*. in *Proceedings Hawaii International Conference on System Sciences*. 1986. Honolulu, HI.
[97] H.A. Sholl and T.L. Booth, *Software Performance Modeling Using Computation Structures.* IEEE Transactions on Software Engineering, 1975. **1**(4).
[98] C.U. Smith, *The Prediction and Evaluation of the Performance of Software from Extended Design Specifications*. 1980, University of Texas.
[99] C.U. Smith. *Software Performance Engineering*. in *Proceedings Computer Measurement Group Conference XII*. 1981.
[100] C.U. Smith. *A Methodology for Predicting the Memory Management Overhead of New Software Systems*. in *Proceedings Hawaii International Conference on System Sciences*. 1982. Honolulu, HI.
[101] C.U. Smith, *Special Issue on Software Performance Engineering*, in *Computer Measurement Group Transactions*. 1985.
[102] C.U. Smith, *Independent General Principles for Constructing Responsive Software Systems.* ACM Transactions on Computer Systems, 1986. **4**(1): p. 1-31.
[103] C.U. Smith, *Applying Synthesis Principles to Create Responsive Software Systems.* IEEE Transactions on Software Engineering, 1988. **14**(10): p. 1394-1408.
[104] C.U. Smith, *Who Uses SPE?* Computer Measurement Group Transactions, 1988: p. 69-75.
[105] C.U. Smith, *Performance Engineering of Software Systems*. 1990, Reading, MA: Addison-Wesley.
[106] C.U. Smith and J.C. Browne. *Performance Specifications and Analysis of Software Designs*. in *Proceedings ACM Sigmetrics Conference on Simulation Measurement and Modeling of Computer Systems*. 1979. Boulder, CO.
[107] C.U. Smith and J.C. Browne, *Aspects of Software Design Analysis: Concurrency and Blocking.* Proceedings ACM Sigmetrics Conference on Simulation Measurement and Modeling of Computer Systems, 1980.
[108] C.U. Smith and J.C. Browne. *Performance Engineering of Software Systems: A Case Study*. in *Proceedings National Computer Conference*. 1982. Houston, TX.
[109] C.U. Smith, et al. *From UML models to software performance results: An SPE process based on XML interchange formats*. in *Proc. 5th Int. Workshop on Software and Performance*. 2005. Palma, Illes Balears, Spain: ACM Press.
[110] C.U. Smith and C.M. Lladó. *Performance Model Interchange Format (PMIF 2.0): XML Definition and Implementation*. in *Proc. 1st Int. Conf. on Quantitative Evaluation of Systems (QEST)*. 2004. Enschede, NL: IEEE Computer Society.

[111] C.U. Smith and D.D. Loendorf. *Performance Analysis of Software for an MIMD Computer*. in *Proceedings ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems*. 1982. Seattle, WA.

[112] C.U. Smith and L.G. Williams, *Software Performance Engineering: A Case Study with Design Comparisons.* IEEE Transactions on Software Engineering, 1993. **19**(7): p. 720-741.

[113] C.U. Smith and L.G. Williams, *A Performance Model Interchange Format*, in *Performance Tools and Model Interchange Formats*, F. Bause and H. Beilner, Editors. 1995, Universität Dortmund, Informatik IV: D-44221 Dortmund, Germany. p. 67-85.

[114] C.U. Smith and L.G. Williams, *Performance Engineering of Object-Oriented Systems with SPEED*, in *Lecture Notes in Computer Science 1245: Computer Performance Evaluation*, M.R.e. al., Editor. 1997, Springer: Berlin, Germany. p. 135-154.

[115] C.U. Smith and L.G. Williams, *Performance Engineering Evaluation of CORBA-based Distributed Systems with SPEED*, in *Lecture Notes in Computer Science*, R. Puigjaner, Editor. 1998, Springer: Berlin, Germany.

[116] C.U. Smith and L.G. Williams. *Performance Engineering Models of CORBA-based Distributed Object Systems*. in *Proc. Computer Measurement Group*. 1998. Anaheim.

[117] C.U. Smith and L.G. Williams. *Performance Models of Distributed System Architectures*. in *Proc. Computer Measurement Group*. 1998. Anaheim.

[118] C.U. Smith and L.G. Williams, *A Performance Model Interchange Format.* Journal of Systems and Software, 1999. **49**(1).

[119] C.U. Smith and L.G. Williams. *Software Performance Antipatterns*. in *Workshop on Software and Performance*. 2000. Ottawa, Canada: ACM Press.

[120] C.U. Smith and L.G. Williams. *SPE Models for Web Applications*. in *Proc. Computer Measurement Group*. 2000. Orlando.

[121] C.U. Smith and L.G. Williams. *New Software Performance Antipatterns: More Ways to Shoot Yourself in the Foot*. in *Proc. CMG*. 2002. Reno, NV.

[122] C.U. Smith and L.G. Williams, *Performance Solutions: A Practical Guide to Creating Responsive, Scalable Software*. 2002: Addison-Wesley.

[123] C.U. Smith and L.G. Williams. *Best Practices for Software Performance Engineering*. in *Proc. CMG*. 2003. Dallas, TX.

[124] C.U. Smith and L.G. Williams. *More New Software Performance Antipatterns: Even More Ways to Shoot Yourself in the Foot*. in *Proc. CMG*. 2003. Dallas, TX.

[125] C.U. Smith and B. Wong. *SPE Evaluation of a Client/Server Application*. in *Proc. Computer Measurement Group*. 1994. Orlando, FL.

[126] C.U. Smith and C.M. Woodside, *Performance Validation at Early Stages of Software Development*, in *System Performance Evaluation: Methodologies and Applications*, E. Gelenbe, Editor. 1999, CRC Press.

[127] SPE·ED, *http://www.spe-ed.com.*

[128] M. Steppler. *Performance Analysis of Communication Systems Formally Specified in SDL*. in *Proc. 1st Int. Workshop on Software and Performance (WOSP98)*. 1998. Santa Fe, NM.

[129] K.S. Trivedi, *Probability and Statistics With Reliability, Queueing, and Computer Science Applications*. 1982, Englewood Cliffs, NJ: Prentice-Hall.

[130] V. Vetland, *Measurement-Based Composite Computational Work Modelling of Software*. 1993, University of Trondheim.

[131] V. Vetland, P. Hughes, and A. Solvberg. *A Composite Modelling Approach to Software Performance Measurement*. in *Proc. ACM Sigmetrics*. 1993. Santa Clara, CA.

[132] L.G. Williams and C.U. Smith. *Information Requirements for Software Performance Engineering*. in *Proceedings 1995 International Conference on Modeling Techniques and Tools for Computer Performance Evaluation*. 1995. Heidelberg, Germany: Springer.

[133] L.G. Williams and C.U. Smith. *PASA^{SM}: A Method for the Performance Assessment of Software Architectures*. in *Proc. 3rd Int. Workshop on Software and Performance*. 2002. Rome, IT: ACM Press.

[134] L.G. Williams and C.U. Smith. *Making the Business Case for Software Performance Engineering*. in *Proc. CMG*. 2003. Dallas, TX.

[135] C.M. Woodside, *Throughput Calculation for Basic Stochastic Rendezvous Networks*. 1986, Carleton University, Ottawa, Canada.

[136] C.M. Woodside, *Throughput Calculation for Basic Stochastic Rendezvous Networks.* Performance Evaluation, 1989. **9**.

[137] C.M. Woodside, et al., *The CAEDE Performance Analysis Tool.* Ada Letters, 1991. **XI**(3).

[138] C.M. Woodside, et al. *A Wideband Approach to Integrating Performance Prediction into a Software Design Environment*. in *Proc. First Int. Workshop on Software and Performance (WOSP98)*. 1998. Santa Fe, NM.

[139] C.M. Woodside, et al., *The Stochastic Rendezvous Network Model for Performance of Synchronous Client-Server-like Distributed Software.* IEEE Trans. Computers, 1995. **44**(1): p. 20-34.

[140] WOSP, *Workshop on Software and Performance*. 1998-2007, ACM Press.

[141] X. Wu and C.M. Woodside. *Performance Modeling from Software Components*. in *Proc. Workshop on Software and Performance*. 2004. Redwood Shores, CA: ACM 488043.

From Annotated Software Designs (UML SPT/MARTE) to Model Formalisms

Murray Woodside

Carleton University, Ottawa K1S 0P7, Canada
cmw@sce.carleton.ca

Abstract. The extraction of a performance model from an annotated software design is largely a matter of taking maximum advantage of the annotations. A serious issue is the fact that a design document directed to producing a product may not be the most convenient for annotation for any given evaluation; there may be a problem to capture the necessary information within the context of the document, without modifying it to clarify the performance concern. Sometimes such a clarification can be of value, but in general we do not wish to disturb the design, just to add the evaluation information. Approaches to using the SPT/MARTE annotations to capture important performance features are described in this paper. Features include *completions* of the design such as platform operations, composition of component submodels, four uses of state machine definitions, and four ways to describe communications costs and delays. The relationship of the annotated design model to the different kinds of performance model is also addressed.

1 Introduction

The evaluation of engineering designs for different kinds of properties is a natural step, but it is only now beginning to be practical for software designs. With the emergence of a widely accepted design notation, in the UML (Unified Modeling language), performance evaluation can be based on annotated UML designs. This has been supported by a standard profile SPT (Schedulability, Performance and Time) [35]. SPT is currently being updated for UML2, and expanded in scope. The planned profile MARTE (Modeling and Analysis of Real-Time and Embedded Systems [36]) will also describe real-time platforms.

In these profiles performance is just one evaluation concern. A second is schedulability, and in future a broad range of evaluations (space, power, reliability, ...) is envisioned. The present discussion is confined to performance concerns.

The general approach to evaluation is to convert the annotated design to a performance model, and to feed the performance results back to the designer, using a tool chain like that shown in Figure 1.

This tutorial is directed towards both performance engineers and software engineers who wish to assist in the evaluation of software designs. It shows how the design annotations can be applied to capture performance concerns, and their relationship to the resulting performance model. It assumes basic background in performance modeling.

M. Bernardo and J. Hillston (Eds.): SFM 2007, LNCS 4486, pp. 429–467, 2007.

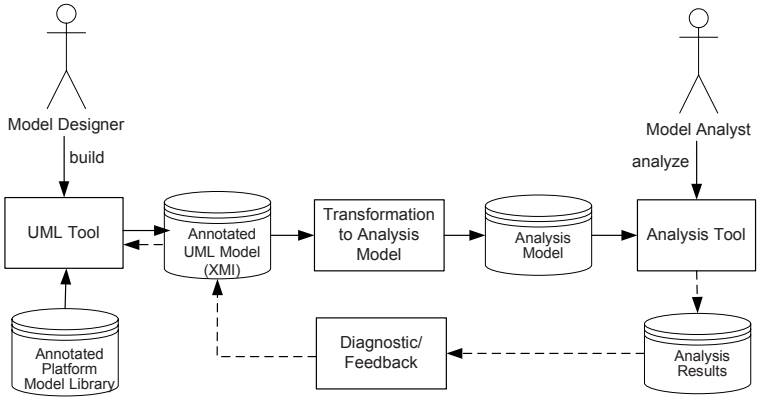


Fig. 1. Tool chain for performance evaluation

We consider evaluation based on a real design model which will quite possibly be used to generate code. There are various ways a design model can be constructed to make performance evaluation easy, particularly by representing the system at the level of abstraction appropriate for the evaluation. However design models are typically at a lower (finer) level of abstraction, and some selection and aggregation is necessary to obtain a performance model that will scale, and that does no need a huge number of parameters. Thus, the software model is not constructed to make evaluation easy. In fact evaluation may be impossible, without adding descriptions of the workload and the execution platform and environment. One of our concerns is to identify what must be added: in general we will call this the *performance completions*.

To be usable, the evaluation method must include:

- performance completions which are easy to add, and do not interfere with the design process
- a performance model used for the evaluation which is produced quickly, without a need for performance expertise in the designer,
- performance results which can be traced back to the design.

This points to automated production of the performance model, as indicated in the architecture of Figure 1.

1.2 Annotations

Annotations are applied as stereotypes to elements of the software specification, that is to diagram elements. The stereotype indicates that the element, which has a role in the functional specification, also has a role in the performance specification. Performance properties are indicated by properties of the stereotype (tagged values in UML1.x), which are written as a property (tag) set equal to a value. Figure 2 shows a stereotype (Pstep) which defines the CPU execution demand (hostDemand) of a function executed by Server.

The annotation is attached to the message, or method call, from Caller to Server; it could equally be applied to the vertical rectangle on the Server lifeline, which is called an ExecutionOccurrenceSpecification (ExecOccSpec) in UML2.

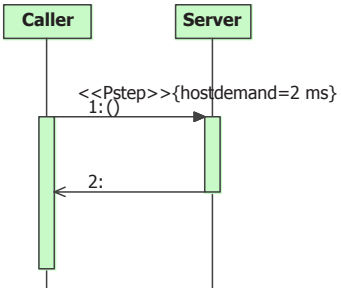


Fig. 2. A performance annotation on a UML sequence diagram

1.3 Purpose of This Article

Our purpose here is to describe how to use the SPT and MARTE profiles for performance evaluation, and to discuss issues:

- the relationship of the design model to the performance model
- annotation for different styles of specification, and different patterns of behaviour
- how to represent important features like communications
- the essential completions needed for evaluation
- alternative cases of the analysis.

For brevity the source design model will often be written DM, and the target performance model, PM.

1.4 A Caveat on the Profile Definitions

The examples in this paper show a lack of consistency in the profile notations used. Examples have been taken from work done over time, during which the standards, and the proposed notations for MARTE, have gradually changed. Thus the performance annotations for SPT were originally prefixed PA when it was first accepted, but they were changed during final revisions to P only, while a new policy at OMG means that in MARTE the prefix will be pa. The MARTE annotations are presently in flux, and examples show considerable differences. It is hoped that the intent will be clear in each case.

This paper concentrates on the principal concepts, and does not provide a complete guide to either profile.

1.5 Literature on Non-UML DMs and Performance

The previous work can be characterized along two axes: the form of the design model (DM), the form of the performance model (PM). We will mention six forms of performance model:

- QN: queueing network
- EQN: extended queueing network
- LQN, layered queueing network, a structured form of EQN

- PN, petri net or similar token-based state model
- SPA, stochastic process algebra
- simulation.

UML is relatively recent as a specification language, and annotated specifications for performance began earlier with the Execution Graph model described in Smith and Williams [45], and its translation into a queueing or extended queueing network model (QN/EQN). Use Case Maps are a general scenario specification technique, to which performance annotations were added by Petriu, Amyot and Woodside [18] with automated creation of performance models described in [17] and a case study of requirements engineering in [39]. A considerable body of work has been done on performance modeling from specifications in SDL, of which representative examples are by Bozga et. al. [6], and Mitschele-Theil and Muller-Clostermann [16].

1.6 Literature for UML and Performance

For UML models, Pooley [21] gave an overview of performance and software in which he described the general concept of annotated UML, with a Petri net PM as an example. Schmietendorf and Dimitrov gave another overview of the issues in [23].

Queueing models (QNs) have been produced mostly from UML Sequence Diagrams. Cortellessa and Mirandola adapted Smith's execution graphs to UML in [8] to generate QNs, and Alsaadi described an application in [1]. An early example of annotations with SPT, with manually derived layered queueing networks (LQNs), is in [20] (Petriu and Woodside). This work showed how to interpret the SPT annotations, and was extended by Xu et. al. to exploit the model for performance improvement in [25]. Sequence Diagram translation was addressed by Kahkipuro, into a layered queueing model [12]. Cortellessa et. al. extended [8] to a complete methodology leading to LQNs in [9]. Automated translation to QNs was described for modeling mobile devices by Pustina et. al. in [22]. Balsamo and Marzolla described the use of translated UML specifications, for architecture evaluation [3].

Automated production of a PN was described Petriu and Shen in [19], using graph grammar tools to produce a layered queueing model from activity diagrams. Conversion of Sequence Diagrams and State Machines to PNs was done by Lo'pez-Grao, Merseguer, and Campos [13] using the compositionality capabilities of labeled Stochastic Petri Nets and by Cavenet et al [7] (to the SPA PEPA-nets) using the compositionality of those nets. Hillston and Wang solved these models by simulation in [10].

Woodside et. al. addressed the problem of translation to multiple types of performance model (QN, LQN, and PN) in [24]. Simulation models which follow the software structure rather than any particular modeling formalism were generated by Bennet et. al. in [4].

1.7 Abbreviations and Acronyms

To make the text less cluttered by long names, a number of acronyms and abbreviations have been used besides DM and PM:

AD	Activity diagram in UML
CSM	Core Scenario Model (sec 2)
DD	Deployment diagram in UML
DM	Design model (e.g. in UML)
ExecOccSpec	ExecutionOccurrenceSpecification (in a SD)
LQ, LQM	Layered queueing, Layered queueing model for performance
MARTE	Modeling and Analysis of Real time and Embedded Systems, project for an extension and update of SPT [36].
NFP	Non-Functional Property (performance properties in MARTE)
PM	Performance Model (a QM, LQM, or PNM)
PN, PNM	Petri net, PN model for performance (stochastic or timed PN), including similar token-based state models
QM	Queueing model for performance
QVT	Queries, Values and Transformations standard [37].
SD	Sequence Diagram in UML, Interaction diagram.
SM	State machine in UML
SPA	Stochastic process algebra PM.
SPT	Schedulability, Performance and Time standard profile [35]
UML	The Unified Modeling Language [41]

2 Relationship of Design Models and Performance Models

2.1 Working from Designer Specifications

A DM is more or less suitable for performance analysis depending on the designer's style and preferences. The coverage of the model that is possible in the PM will depend to some extent on the richness of the DM. Adaptation to a less-than-ideal DM is part of the use of the annotations.

2.1.1 First: Components and Code

A bare-bones software design must at least define the components and their interfaces in a Class diagram (Figure 3a). Then code fills in the functional details to create the product. Behaviour, including the relationships between components, is expressed only in the code. A performance model cannot be created from such a design definition; it must first be enriched in various ways. (This is in fact a significant barrier to evaluation.) A performance evaluation requires further enrichments, as well as quantitative annotations as illustrated in Figure 2.

The main annotations are

- resource demands (execution resources, called hostDemands, and demands for operations by non-CPU resources), and
- behaviour prameters (loop counts and probabilities of sub-paths)

A bare-bones Class design can be annotated on the operations of the classes, so show how one operation calls another. By itself this does not show how different instances of the class behave, so it is only adequate if all instances require the same annotations.

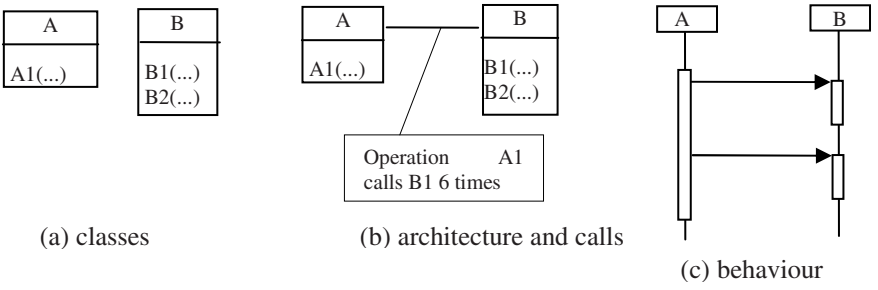


Fig. 3. Design models of varying richness

2.1.2 Enrichment (1): Add Architecture (Figure 3b)

An architecture model adds relationships between components (e.g. box-connector diagrams). The connectors may identify multiple kinds of interactions between the same pair of components. This can be described in UML by a CompositeStructure diagram with the component notation of required and provided interfaces.

The different kinds of invocation of a component (say, component B) may trigger quite different behaviour. We will call these *services*. An example is the invocation of a read on a database, versus an update. Each kind of invocation of component B will be said to trigger a corresponding *service* (say, service B1) by the component. An additional enrichment of the architecture definition is to identify the further interactions that are part of service A1, with other components and services. Such a definition gives a call hierarchy of services in the system.

Again, annotations applied to a CompositeStructure diagram apply to classes, with the restriction that they imply that they have the same values for all instances. To deal directly with instances we must consider UML Collaborations, in which instances take roles and execute behaviour. Collaboration diagrams do not show operations, so annotations must be applied to behaviour.

2.1.3 Enrichment (2): Add Behavior (Figure 3c)

Designers often specify the significant behaviour of the system by three kinds of UML diagrams. Interactions (as in Figure 2c) show the sequence of interactions and operations in a set of collaborating roles. Activities show sequences in a different way, and are meant to be more abstract. State Machines show the behaviour of a single class and define the part taken by one participant. All three of these behaviour styles may be interpreted as Scenarios in SPT and MARTE, so we will consider Scenarios.

In a Scenario, a Step is an action by the system, possibly just an execution on a host. A Scenario is a sequence of steps with a single starting point, generalized to include branching and forking of the sequence. A Step can be refined as a Scenario. A primitive Step (called in MARTE an ExecStep) executes on a *host* processor called an ExecHost, which must be identified by deployment of the process which executes the Step. The annotations will be treated next.

2.1.4 Enrichment (3): Add Execution Resources and Workload

The execution resources are processors which run the software processes, which execute the Steps. Processors are identified by deployment.

Deployment is a major source of variation in software designs, and evaluation must often cover multiple deployments. In that case the binding of software components to processors must be parameterized, and is better done outside the DM.

The execution of code on a host processors is given by its *hostDemand*, usually stated in terms of processing time. If we know the host demands of the Steps, we can derive the host demands of a single execution of a Scenario by addition.

The Workload is a completion that is not part of the design, but is a part of the evaluation. The commonest forms are an arrival process (open arrivals) at a given rate, or a closed workload with a given population of users.

2.1.5 Enrichment (4): Add Logical Resources

"Logical resources" in this tutorial will stand for a wide range of software mechanisms that are used to protect execution paths and storage operations. These resources will be thought of as logical tokens that must be obtained by the program in order to proceed with an operation. Examples of logical resources include:

- a process thread pool is a resource with multiplicity equal to the pool size.
- buffer pool
- semaphore or mutex, usually a resource of multiplicity 1.
- admission control token pool.
- lock. An exclusive lock is a resource of multiplicity 1.

From our point of view a logical resource is any mechanism which can delay the program in order to allocate a token to it, based on the availability of tokens which are neither created nor destroyed. The delay may involve queueing, or retrying by the program. A process resource is a particularly common form of resource, and all execution is done in the context of a process.

A scenario with logical resources can be represented by a scenario metamodel for performance, called the Core Scenario Model (CSM) [40]. Steps are represented in sequence, with connectors for branch (or-fork), merge (or-join), fork and join. Figure 4 shows a CSM model for a specification that will be described later. The Steps are rectangles, linked by sequential connectors which are fat arrows, and there is one fork connector. Resource acquisition and release steps are explicitly placed in the scenario, and associated with their resources. Four kinds of resources are shown in Figure 4: Processes, (shown as squares labeled "Component"), a passive resource "Buffer" (a rounded square), an external operation resource (to be discussed), and host processors (circles). Host processors are not acquired and released explicitly because that is too fine-grained a behaviour to model, but each Step is associated to a host through its Component.

The acquisition and release of resources must satisfy certain constraints when paths branch or fork. When they branch, only one path is taken so each path begins with the same resource context, however the different branches may change the context in different ways by acquiring and releasing resources. The resources held on all subpaths should be the same at the following merge, however.

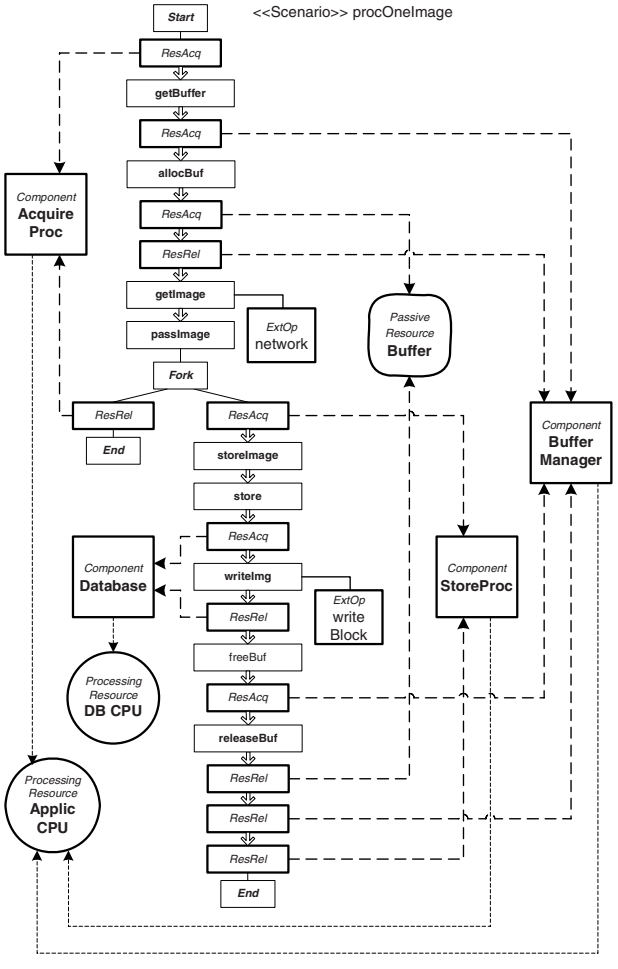


Fig. 4. The Core Scenario Model for the Video Capture Scenario discussed below [40]

At a fork, all the successor paths are taken concurrently, and to retain exclusive access a resource must be explicitly passed to one of the paths. At a fork, the current process resource is forked also, and each branch obtains a copy without (by assumption) having to acquire it by a request.

At an end-point of a system scenario, all resources are released. However at an end-point of a nested behaviour, nested for instance within a Step, they are retained and kept for the outer scenario.

2.1.5.1 Nested pattern of resource usage. Resources can be acquired and released in any order. Nonetheless, it is well known that acquiring multiple resources in an order prescribed by a global ordering of resources, and releasing them in the reverse order to their acquisition, avoids resource deadlock (for locks, this is called a “global

locking order"). The result is what we will call nested resource usage, since for any pair of resources held simultaneously one is acquired earlier and released later, than the other. This is illustrated in Figure 5.

In the nested case the *resource context* (the set of resources held by the program at any point in time) grows and shrinks in a particular way. If the resources in the context are ordered in the order they were acquired, it is always the last one that is released first.

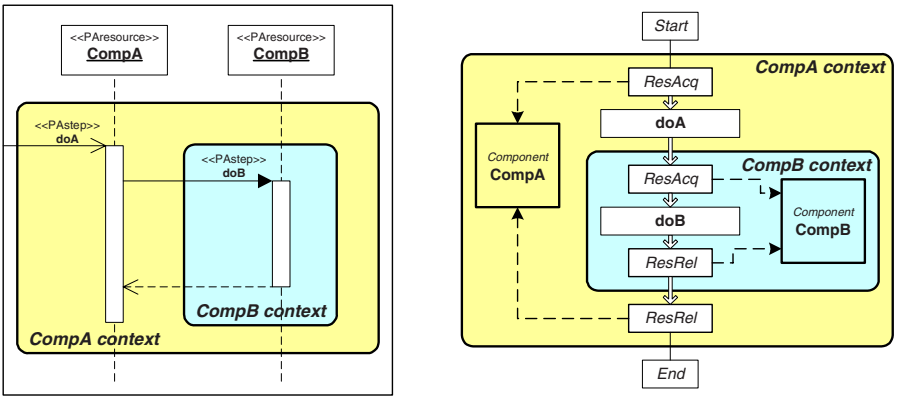


Fig. 5. Illustration of resource context: sequence diagram on left, CSM on right [40]

The global ordering referred to above ensures that all of a concurrent program's ordering of acquisition are compatible. This order also establishes an order of layers of resources that can be used to construct a Layered Queueing Model (LQM). This is a special simple canonical form of Extended Queueing Network, which can be solved by analytic approximations that take advantage of the layered structure (or equivalently, of the nested resource behaviour).

2.1.5.2 Logical Resources and Extended Queueing (EQNs). Extended QNs (see, e.g. [27][33]) are approximations for systems without exact solutions. Logical resources map directly to servers in EQN which provide simultaneous Resource Possession.

When resources are acquired and released in an arbitrary order, resource deadlock can occur. Extended queueing approximations can be constructed as described in [45], or a network model can be applied. Petri Net models, for instance, represent the states of execution and resources directly by tokens for each, and can represent any resource behaviour whatsoever. They can also be checked for deadlock. The choice of PM is discussed further below.

2.1.5.3 Nested Resource Usage and Layered Queueing. Layered queueing [30] [31] [32][42] is a form of EQN in which the acquiring and release of resources is partially nested [51]. It compactly describes the interaction of many resources held in many different combinations, and has been applied to enterprise systems [40] and embedded systems [33].

LQN model concepts are illustrated in Figure 6. Bold rectangles for WebServer, Database, Disk, and Net represent logical resources called “tasks” (labeled by a multiplicity), with attached rectangles for classes of service called “entries” (labeled with their CPU demand and an optional pure delay). Workload is initiated by the task User which cycles forever (implicit in that it receives no requests). Entries make requests to other entries, indicated by arrows labeled by mean numbers of requests. A blocking request (in which the requesting task or thread waits for the reply) is indicated by a solid arrowhead, and all requests in Figure 6 are blocking. Asynchronous requests are also possible. Tasks are hosted by devices indicated by an ellipse. Each task and processor has a queue. As with other extended queueing models, an LQN is solved by decomposing it into a number of queueing networks and iterating between them, or by simulation.

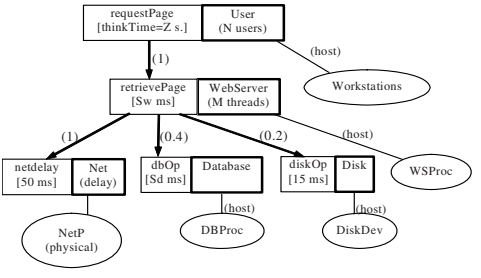


Fig. 6. A Layered Queuing Model of a Web application ***track06j

In Figure 6 the User task represents N separate users and their browsers, each of which alternately sends a requests to the Web Server every Z s and waits for a response. The WebServer represents the server software, including the application, with M threads and processor WSProc. The entry RetrievePage serves the users, with CPU demand S_w ms, one network latency of 50 ms, and on average 0.4 database operations and 0.2 disk operations. The disk and the database are here defined as single servers with a queue, running on their own devices DBProc and DiskDev, with CPU demands of S_d and 15 ms, respectively.

2.1.6 Enrichment (5): Add Completions

Completions is a term used in [53] to describe elements that must be added to a software design to make it possible to evaluate it for performance. It includes the resource and workload related annotations described above, but also:

- components or subsystems that provide system services that are invoked explicitly, such as database services, file services, directory services, and web services.
- services that are executed by the software platform and not invoked explicitly. The software platform may include the operating system and network protocols, middleware, and network services,
- indications which imply elements of either of the first two categories.

In a model-driven development approach, policies govern transformations to include platform elements in a design, and these policies could (in principle at least) be used to generate the corresponding completions.

Completions can be incorporated into the analysis in two ways, essentially either by adding them to the UML model, or by adding a performance submodel that represents the effect of the completion. Both are useful. It is a question of composing a platform submodel which may be available at one level, or the other. Creating performance submodels for platform components relieves the effort and complexity at the UML level but reduces the flexibility to adjust the submodel to the system.

A performance submodel is identified in the annotations by indicating requests made to an *external operation*, and these are described in section 3.7 below.

2.2 Measures for Performance Quantities

It is unusual for any kind of quantitative evaluation of a system to be done for single set of parameter values. Sensitivity to its parameters is one of the properties that is studied in an evaluation, and a great number of parameters may be varied. Thus it is essential to think of the performance analysis as being parameterized with variables for the parameters, rather than with point values.

Performance measures for a system take a number of forms which are sometimes not clearly separated. Consider a response time for a certain operation:

- the abstract definition of the response time measure identifies it with a certain start event and a certain end event, which begin and end the response. It has a name, say HomePageResponseTime.
- there is a *manifestation* of the response time during the running of the system, which might be captured by instrumentation and recorded in a trace. this might be the value for the 31st response in a certain trace (say, trace T71).
- there is a *realized statistical property* of the measure, over manifestations, such as the average value for trace T71, of the maximum value in the trace.
- there is an expected value for HomePageResponseTime over manifestations, treating it as a random variable which occurs in the system. We will call this a *theoretical statistical property*. for the system. The variance is another such measure, the 95th percentile is another.
    - Note that the theoretical maximum value (if it exists) is the largest value that can ever occur.
- there is another expected value for it as a random variable in a model of the system, which we will call a *model statistical property*.

In performance analysis we are interested in various *statistical properties* rather than in manifestations.

We may need to consider both model values and measured values; we will describe these by the *source* of the measure. Other sources for values may be assumptions, and requirements. To summarize, a performance measure has two modifiers, its source and its statistical property, with typical values:

source = { req (requirements) | est (estimation-by-model) | asmd (assumed) | meas (measurement) }
statistical property = {mean | variance | stdDeviation | percentile(level:[0..100]) | max | median.....}

To confuse things a little more, the same measure may also be described by a probability distribution (in theory, assumption or requirements) or by a histogram (found by measurement). One way to express this is to extend the list of statistical properties to include:

... | distribution(shape, parameters) }
shape = {histogram | exp | erlang | gamma | uniform | normal |....}

The expression in SPT and MARTE follows these principles, but differ in the details.

2.3 Design Abstractions and Performance Abstractions

A performance model (PM) essentially describes how the execution of the design uses the system resources, at some level of abstraction. Based on the behaviour and execution resource demands a PM can be created, representing the state of execution of a single instance (which operation is being performed) and its transitions, and the time the operations take. With a workload definition, concurrent execution introduces contention at the hosts, and a queueing model follows, which represent the state not only of the execution, but of its resource requests (waiting, using). With additional resources, an extended or layered queueing model is necessary. In the most general case it may be preferred to use a Petri Net model which can represent arbitrary combinations of execution state and resource state, and arbitrary resource operations. The PM abstractions thus are effectively aligned with the DM level of detail.

From the above we see that the process of extracting a performance model can be reduced to two problems:

- identify the operations with their host demands and sequence constraints,
- identify their use of other resources, if appropriate.

We will sketch in the relationship between the Scenario in the DM, and the constructs of the different PMs.

2.3.1 QN from Annotations

An ordinary QN represents requests for service by servers which correspond to processors or other hardware devices. The model is determined by the set of servers and the total demand for each server, plus the workload description for the arrivals.

The processor demand parameters are calculated from the hostDemands of Steps. Some Steps are refined into sub-scenarios; only primitive Steps with no refinement (ExecSteps in MARTE) have processor demands. Suppose the ith ExecStep has host $j=h_i$ and hostDemand d_i. Then the set of total hostDemands $D_{k,j}$ for host j in any segment k of the scenario is found by:

- from the behaviour, find y_i = mean number of times ExecSTep i is traversed in the segment, for all i
- $D_{k,j} = \sum_{\{i \mid \text{step i in segment k, } j = hi\}} d_i$ (Equation 1)

For a QN model, the Scenario segment k is the entire scenario, $D_{Scenario,j}$ is the demand for the jth server, and the above calculation has a similar effect to the one used by Smith and Williams for reducing execution graphs in [45]. Further, if there are multiple scenarios to be combined in the operation of the system, two QN cases may occur:

- each scenario that has a separate Workload becomes a distinct Chain in the model, with its own user population (class of users).
- where scenarios form alternative behaviours initiated by a single Workload, they are combined as alternative paths within a single overall system-level Scenario. They can be modeled as separate classes of behaviour within a single chain.

Multiple Scenarios which are initiated by a single class of users but are triggered in some particular sequence governed by a State Machine can be combined using the concept of a WorkloadGenerator, described below. The probabilistic path of the SM is first reduced to a mean number of invocations of each component Scenario, per client cycle, and they are then combined using the second option above to give a single composite response.

2.3.2 LQN from Annotations and Structure

For a LQN model, the structure is more complex. Each software process becomes a task, with entries for the operations carried out the process. The calls between entries are identified from the transitions between processes in the Scenario.

HostDemands are totaled by process. For each process a Scenario segment in the sense of Equation 1 is determined by examining the resource context of the Steps. It includes those Steps with a resource context corresponding to execution of the process, which becomes a task in the LQM.

Multiple top-level scenarios are treated similarly to the QN construction above. Scenarios with separate workloads each become a separate driver "task", those which are alternatives for the same workload are combined as alternative requests from a single driver task.

2.3.3 PN from Annotations and Structure

A PN model constructs places with token pools to represent sources of requests such as users, and logical and physical resources. Each Scenario is represented as a sequence of PN places and transitions for the Steps, with transitions enabled by the user and by resource tokens where resources are acquired and released.

3 Basic Annotated UML Interactions (Sequence Diagrams)

We will now tour the performance annotations of SPT and MARTE in more detail. As in Figure 2, annotations take the form of stereotypes applied to software design elements. The version of MARTE annotations shown here is preliminary and by the

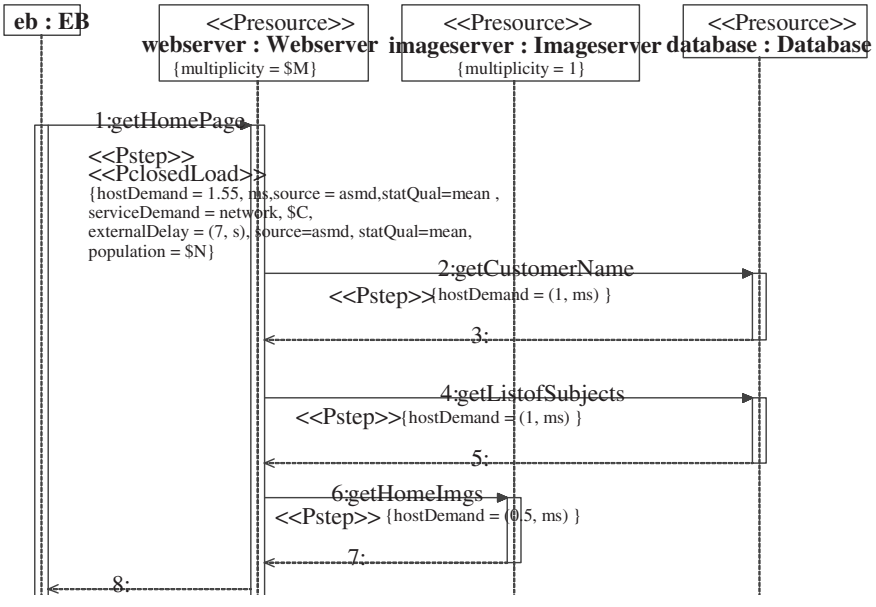


Fig. 7. A simple interaction diagram with annotations from SPT

time you read this it is probably incorrect in details, but the concepts behind the annotations are mostly stable. Thus, this is not a guide to the precise use of MARTE, but to its concepts.

3.1 SD with Workload (SPT)

Figure 7 shows a sequence diagram for a user (represented by the emulated browser eb) interacting with a web server, which executes an application requiring database access. This is a simplified version of one of the interactions in the TPC-W benchmark specification, representing an on-line bookstore. It is annotated with stereotypes from SPT.

In a SD, the interacting elements are shown by lifelines (the vertical dashed lines, labelled by the system element). Here they represent interacting processes which are stereotyped as resources (Presource). The execution of operations is indicated by the vertical rectangles over the lifelines, called in UML2 an ExecutionOccurrence-Specification (which we will abbreviate to *ExOccSpec*).

3.2 Combined Fragments

UML2 provides “CombinedFragments” for combining subscenarios in many ways, e.g. as alternatives, in parallel, for a loop, and for a reference to an imported subscenario. Figure 8 shows the same scenario as Figure 7 with optional execution of the first part, and inclusion of a subscenario for a special promotion which is part of the benchmark. SPT annotations are still used although they do not strictly speaking apply to UML2.

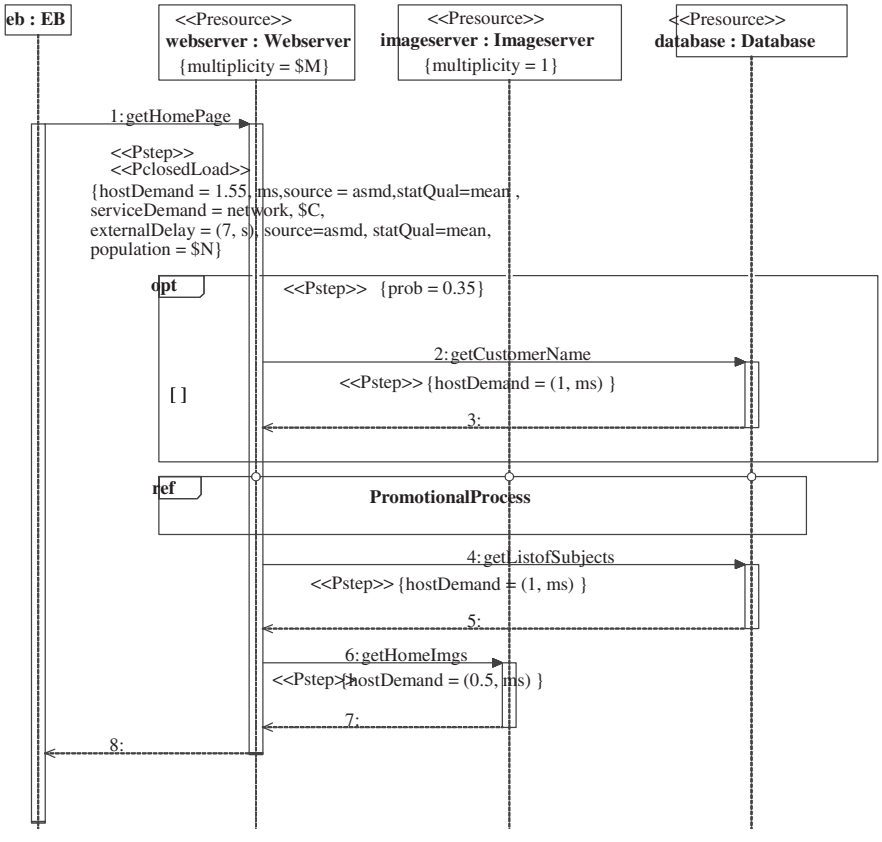


Fig. 8. The GetHomePage scenario for the TPC-W specification (electronic bookstore) (as Figure 7, adding UML2 CombinedFragments for opt and ref)

The argument of CombinedFragment is the portion of the behaviour which appears within it, and this can be stereotyped as a Step. Important types of fragment for performance annotation include:

- `opt`, with a probability as shown
- `alt`, with Step with its own probability for each alternative part
- `par`, for Steps in parallel
- `loop`, with a repetition count for the Step
- `ref`, for a sub-scenario defined by another behavior (it may also have a repetition count if repeated)

A step in parallel may or may not participate in the joining of the parallel paths, at the end of the `par` block. UML lacks a notation for this, so a special boolean attribute "noSync" is applied to arguments of `par`, which do not join.

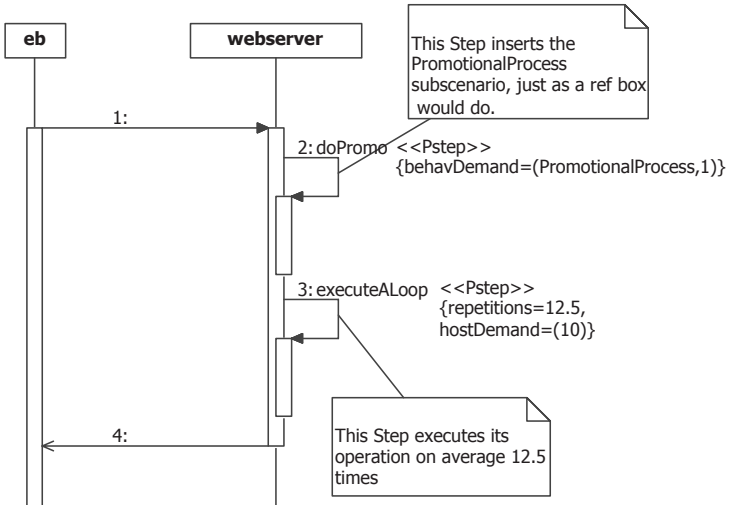


Fig. 9. An explicit demand for an inserted subscenario, and illustration of repetition by annotation

3.2.1 Explicit Demand for a Subscenario

Sometimes a sub-scenario is needed when a `ref` block is not appropriate. An example which will be discussed further below is, to provide a sub-scenario for communicating messages between two processes on different nodes. Behaviour which is included in many places is efficiently captured this way.

Figure 9 shows a reference to a sub-scenario within a Step, as a demand for inclusion of a behaviour. The demand can have a count parameter to describe repeated behaviour. It is intended to have the same semantics as a `ref` block, as far as inclusion within the scenario is concerned.

3.3 Explicit Logical Resource Operations

Logical resources representing software processes are implicit in the DM, but some logical resources must be represented explicitly, with their points of acquisition and release. An example is a buffer from a pool shared by concurrent threads or processes, managed by a concurrent buffer manager, and illustrated in Figure 10. The buffer manager is a logical process resource which must not be confused with the buffer pool. When the request returns from the manager it releases the manager resource but retains the buffer resource, as illustrated in Figure 11.

To represent explicit acquisition and release, the operation is stereotyped to identify the resource and the number of units of the resource (such as the number of buffers requested). SPT provided annotations GRMacquire and GRMrelease, shown in the Figures, and MARTE has subtypes of Step paAcqStep and paRelStep to do the same. If they are applied to a behaviour entity which is also an ExecStep, an acquisition occurs before the ExecStep, and an release occurs after.

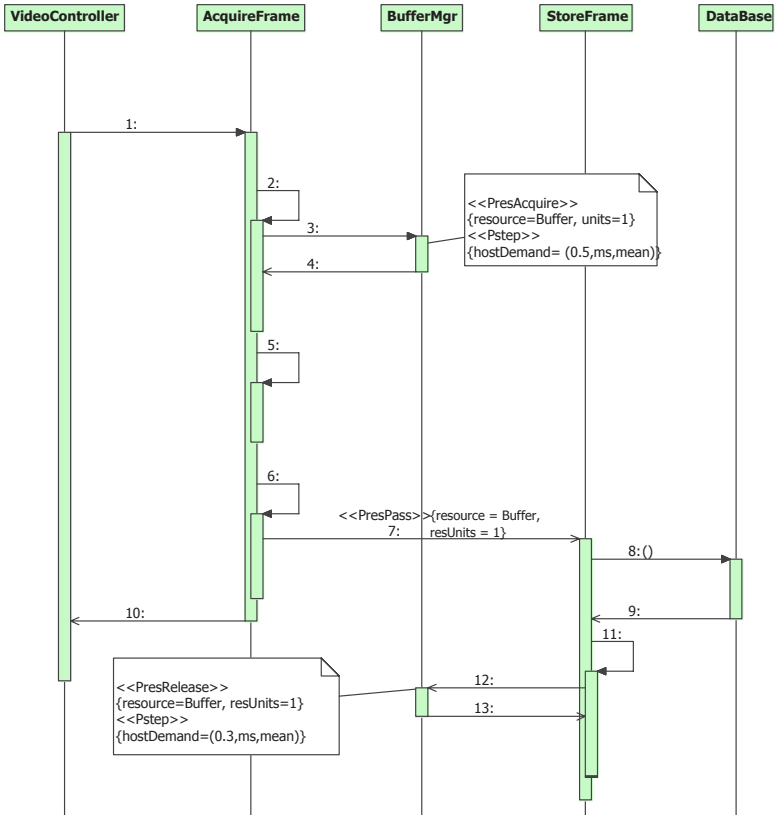


Fig. 10. A SD showing explicit acquisition, passing and release of a resource in a buffer retrieval/storage system

An additional operation that affects resource contexts is passing a resource from one process to another. In order to capture this in all behaviour diagrams a stereotype <<paResPassStep>> is included in MARTE, also identifying the resource and the units. It is illustrated in Figure 10.

The logical resource itself needs to be an object identified as a Resource with its units as a multiplicity attribute.

3.4 Demand for "Services" Offered by a Separate Submodel

To support component-based design, we wish to assemble a system scenario from already-developed subscenarios for a set of components. The components may either be product elements used by a "glue" program, or platform components incorporated in a transformation to a platform-dependent model (PIM to PDM) in model-driven development. In either case we assume the components already have annotated scenarios, connected to interface operations of the component. We will refer to the operations as its *services*.

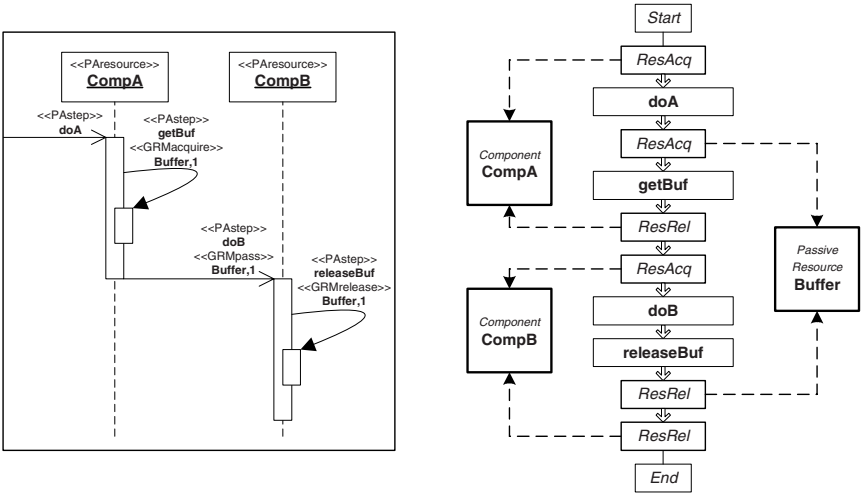


Fig. 11. Acquisition and release of resources, and resource contexts in CSM [40]

In MARTE, to connect the interface operation to a defining scenario, it is stereotyped <<RequestedService>> with a property behaviorDefinition set to the defining scenario. The invocation of the interface operation by a Step is represented by a Step annotation servDemand:RequestedService, and servCount:Real for the mean number of operations. Figure 12 illustrates the use of a service of a component by a Step.

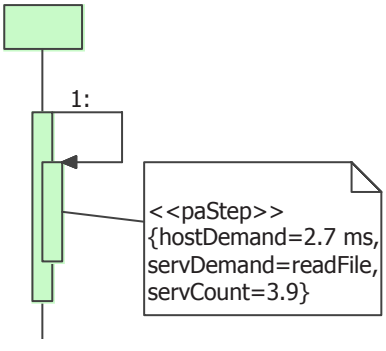


Fig. 12. Step that uses a service readFile of a component FileSystem

3.5 Service Hierarchy Annotation

A subsystem that does not have any behavior specifications can still be annotated (proposed for MARTE) at the interfaces of components, effectively creating a graph of calls, with a workload for each operation. The Layered Queueing Network model illustrated in Figure 6 is a kind of representation for this graph. Each operation executes some host computation and makes some calls to other operations, and

quantities for these are annotated on the operation. Each operation can have the properties of a Step, including

- a host demand
- a set of demands for other operations
- a set of demands for external operations
- a set of demands for included sub-scenarios.

Figure 13 shows an example for a FileSystem which invokes operations from two sub-processes Cache managing a cache, and FileServer managing the storage subsystem. The servDemand and servCount attributes are ordered to correspond, so the attributes mean that each readFile operation invokes one read from the cache, and in the case of a cache miss (specified as 3.5% probability) it also reads from the storage system.

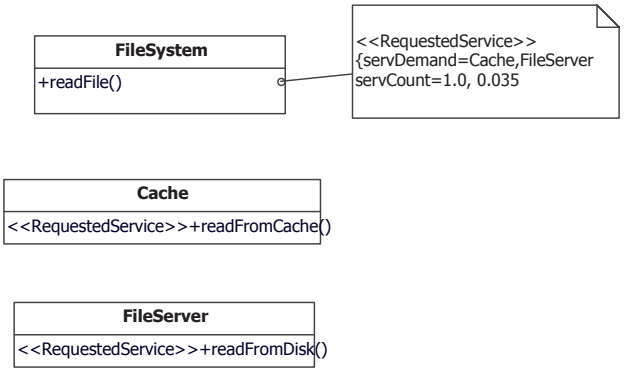


Fig. 13. Representing a call hierarchy within a component subsystem (in MARTE)

With these annotations, any pattern of behaviours invoking services, services invoking services and services invoking behaviours can be captured.

3.6 Entirely Structural Model

Annotated operations can be applied to an entire system design which does not have behavior specifications. The workload can be attached to the top-level operation, and annotations can describe the call hierarchy between objects annotated on class and composite structure diagrams, with call behavior and execution demands for operations.

There are limitations in annotating the classes, in that by default the annotations apply to all instantiations of the class. These may be different depending on the configuration of the instances, which UML accounts for by describing behaviour in different collaborations. Instances may also have different deployments. An approach to support annotation of class descriptions, by adding configuration and deployment information to the translation, is described below in Section 4.3. Here we shall assume that values can be resolved for instances as described there, and describe the conceptual basis of the annotations, and the production of performance models, on the assumption that.

It may be more difficult to estimate numbers within this approach because the number of demands for lower level services must be aggregated into one demand figure. This is exactly the role of the scenario analysis, to assist in the analysis of these demand counts. Nonetheless, if one can come up with the demands from one operation to another, the description can be constructed and the performance model can be extracted. And the demands correspond to behaviour of units of code which can be measured by profiling, which may assist capturing numbers for library subsystems.

3.6.1 Queueing Model from Structural Parameters

A queueing model is created by aggregating all the demands for each host. Consider RequestedService (operation) RS_k, which has host demand d_k and makes service demands. For the demand to $RS_{k'}$, we denote the servCount parameter (the mean number of requests) as $y_{k\,k'}$.

Denote by D_{kj} the total demand for host j created by one request to RS_k. It is therefore the QN server demand, per invocation of RS_k. If a class of users invokes RS_k, these are the demand figures for that class in the QN model.

Provided there are no loops in the recursion, D_{kj} can be found by summing over the requests made by RS_k, and computing the sum recursively over all RS_k in the specification:

$$D_{kj} = [\text{if}(h_k = j)]\, d_k \; + \; S\, k' \; y_{k\,k'}\, D_{k'j} \qquad \text{(Equation 2)}$$

3.6.2 Layered Queueing Model

A LQN has a more direct relationship to the parameters of the RequestedServices. Each active object becomes a task, and each of its operations becomes an entry in the model. If all the annotations are operations of active objects, then the entry parameters for RS_k are:

entry host demand = h_k
entry call demands to the entry for $RS_{k'} = y_{k\,k'}$

If part of the hierarchy belongs to a single process and RS_k is one of the interface operations of that process, then for RS_k the entry host demand is found by applying Equation 2 above to the operations by objects within the same process only, and the entry call demands are found as follows.

Let RS_i be an interface operation of some other process, and let Y_{ki} denote the mean total calls to RS_i for each invocation of RS_k. Again assuming there are no loops in the recursion, Y_{ki} is found by summing the calls over the operations invoked by RS_k, and computing the recursive values, as follows:

$$Y_{ki} = y_{ki} + \Sigma_{k' \text{ in the same process}}\, y_{kk'}\, y_{k'i} \qquad \text{(Equation 3)}$$

3.7 External Operations

The need for external operations to represent platform components was introduced in Sec 2.1.6 above. They are operations which are not described in detail in the UML DM, but whose details have performance impact.

Table 1. Preliminary summary of annotations for this section

Annotation	Meaning	Performance Model Significance
«PClosedLoad» population, paRequestEventStream population	concurrency of the workload	population of a class of customers (QM, LQM); size of a customer token pool (PN)
PClosedLoad externalDelay, paRequestEventStream externalDelay	delay at the user	think time
OpenLoad arrivalRate, paRequestEventStream arrivalRate	rate	arrival rate
Pscenario, gaBehaviorScenario	top level: a system response nested: a structured unit of behaviour	top level: behaviour of a class of customers nested: to be flattened into the total behaviour.
Pstep, paStep	a part of a response	defines part of the behaviour
hostDemand of a Step	execution requirement	an increment to a device service time (for the host device for that Step) (QM, LQM
Phost, paExecHost	the Node which executes some Steps	a server (QM); a processor resource (LQM); a resource token (PNM).
Presource, grResource	a logical resource of any kind	a simultaneously-held resource (EQM); a task or software resource (LQM); a resource token (PNM)
paProcess	a logical resource for a process thread pool	as for Presource etc.
paLogicalResource	a logical resource other than a process thread pool	as for Presource etc.
paRequestedService	a hierarchical unit of behavior associated with a subsystem and a separate specification	part of the workload to be executed.

This is a practical issue, which is manifested in well-developed modeling frameworks in libraries of submodels for well-known and often-reused components. In an environment for modeling enterprise systems, there could be submodels for database systems and commercial storage products, with parameters for their

configuration and for the way the subsystem is to be used (the database size, for instance). Having the submodel takes some of the effort out of modeling, and reuses important knowledge.

A very relevant example is a model which captures the arcane details of the TCP protocol, which no one would wish to model in UML. An example of a LQM performance model coupled to a network simulation model is given in [50] (although it is not coupled using the profile annotation, but in the modeling environment).

The external operations annotations in SPT and MARTE provide hooks directly to submodels such as these, by name. The annotations take the form of two attributes, as for a service demand: extOperation:String and opCount:Real). The annotation does not fully define the submodel, so the semantics of the included operation are left to the performance environment.

3.8 Summary of Annotations and Their Significance

To clarify the use of different annotations from SPT and MARTE in this section, they are summarized to in the following table. The prefix P is from SPT, and is sometimes seen as PA; the prefix pa, ga or gr is from MARTE.

4 Parameterization

4.1 Input and Output Properties and Parameters

There are two kinds of quantities in any evaluation, inputs and outputs to the evaluation. A quantity identified as a performance measure in the annotations may be of either type. In general the input measures are:

- behaviour parameters such as probabilities, and repetitions of Steps.
- demand parameters both for host demands and demands for services and operations.
- required values of performance measures such as delay and throughput.

The outputs are performance measures estimated by the PM. The same quantity may appear with a required value as an input and an estimated value as an output, along with a judgment as to meeting the requirement.

Parameters of an evaluation are not the same thing as input measures to the evaluation. Parameters are *variables* which govern the measures through expressions. When values are given for the input parameters, the input measures are determined, the evaluation can be carried out, and output measures are returned.

4.2 Variables and Relationships

It is not enough that an annotated value can take a symbolic value as a variable, it must also be possible to establish functional relationships. For example, several demand parameters may depend on a single parameter of an operation, such as the size of a data structure to be processed. We wish to specify one value for the size and have all the demand parameters adjust according to the relationship.

For an example we shall use the buffer capture/storage system sketched earlier in Figure 10, and repeated in Figure 14 with more extensive annotations [25]. There are five processes. VideoController cycles through the list of cameras, and in turn sends a procOneImage request to AcquireProc, a process which acquires one frame from the next camera in the cycle. AcquireProc gets a buffer from the BufferManager, fills it, and passes it to another process StoreProc which stores it in the database. Using a concurrent storage process is a performance optimization allowing overlap of the acquire and store operations; once AcquireProc has passed the buffer it goes to the next camera in the cycle. The system is thus self-timed, and the requirement is that it complete the cycle in one second, for 95% of cycles. There will be variations in operation times because the size of a frame (compressed) varies. The explicit passing of the buffer resource, which is shown in figure 10, is not shown here because it was not part of SPT.

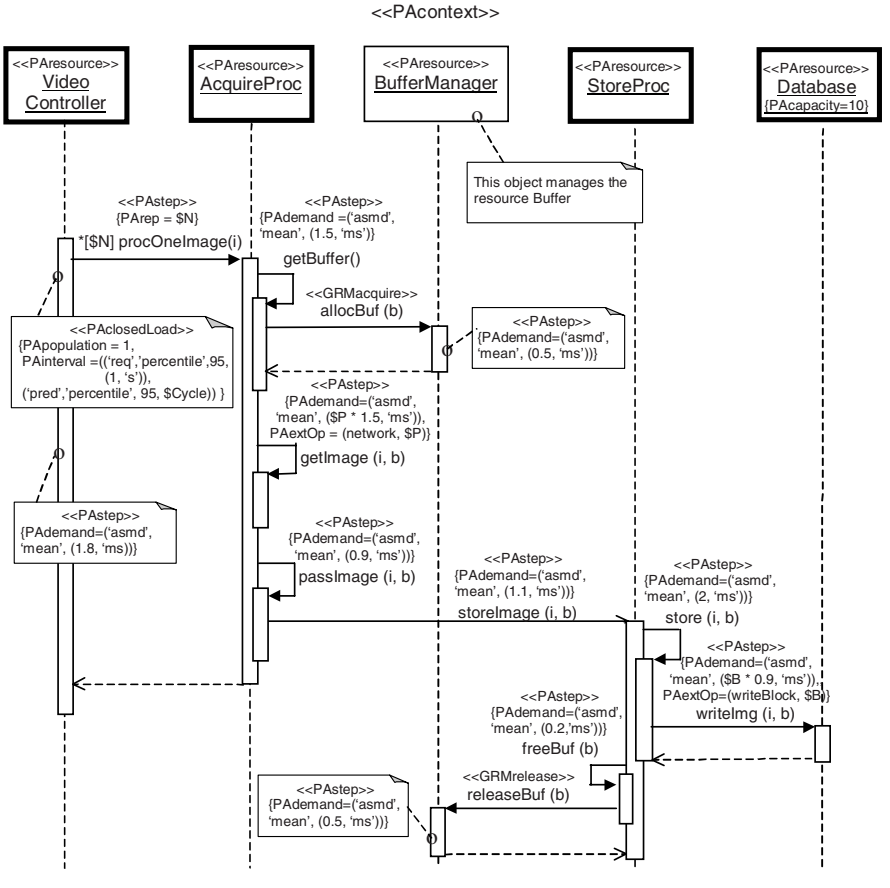


Fig. 14. The video buffer capture/store scenario from [25], Figure 4

In Figure 14 there are four parameters, shown with the prefix $ for parameters:

- $N is the number of cameras
- $Cycle is the time to poll all the cameras in a cycle,
- $P is the number of network packets required to transmit one frame (one video image),
- $B is the number of blocks on disk to store one frame.

There is also a parameter in the deployment diagram Figure 16 below,

- $Nbuf, the number of buffers in the buffer pool.

The network operation for the frame is annotated as an external operation, carried out $P times, once per packet. The execution demand for the capture operation depends on $P (hostDemand, called here PAdemand, is 1.5* $P) and the execution demand for the storage operation depends on $B.

The performance evaluation of this system reported in [25]examined the role of $Nbuf, and some features of the software design, in meeting the specification on $Cycle with the largest number $N of cameras possible. The specification is shown as:

{PAinterval = ('req','percentile',95, (1,'s'))

which means the interval between successive initiations of the whole scenario (meaning, a cycle of all cameras) is required to be less than 1 second, in 95% of instances. Thus every camera is polled once every second, with 95% confidence.

The performance evaluation found that with the design shown, asynchronous handoff of the buffer, and multi-threaded processes, 40 cameras could be supported and the limiting issue was the response time of the AccessControl rather than the VideoAcquisition.

4.3 Parameters for Instances

Certain UML types can only be annotated for classes and not for instances. In general UML assumes that it is classes that are being designed, and instances are only studied in typical behaviour and in collaborations.

- Annotations on classes apply by default to all instances.
- Annotations on behaviour apply to all instances that implement that behaviour (they apply to roles, not instances).

Thus, if there may be multiple instances of a subsystem that implement the same classes (in the first case) or behaviours (in the second case), the UML annotation cannot differentiate them. Here we consider the case where they need to be differentiated, and suggest a workaround based on using variables for all the affected parameters, and instance values defined separately.

The idea is quite simple: use variables or parameters that define the hostDemand, repetitions, probabilities, message sizes and other attributes that vary among instances, and tabulate the instances and their deployments outside the UML model, with concrete values for the variables. We can distinguish two cases.

If the only difference is in the deployment and the speed factors (relRate) of the hosts, then the same behaviour diagram can be used for all, and only the deployment and rate information differs. Suppose that, in the video buffer capture system of Figure 14 we wish to have

- three separate video systems are to share the same database system, and
- the process instances for VideoController, AcquireProc and StoreProc are subscripted 1,2,3,
- they are all deployed on the same processor, for each subsystem.

The nominal deployment of a single system is shown in Figure 16 below. Then for the three subsystems, the processor demands will be divided by the relRate values for the deployed processor. A table like the following could define the parameter values.

Process	Host, relRate		
	Video1	Video2	Video3
VideoController	PC1, 1.0	PC1, 1.0	PC3, 1.26
AcquireProc	PC1, 1.0	PC2, 1.74	PC3, 1.26
StoreProc	PC1, 1.0	PC2, 1.74	PC3, 1.26

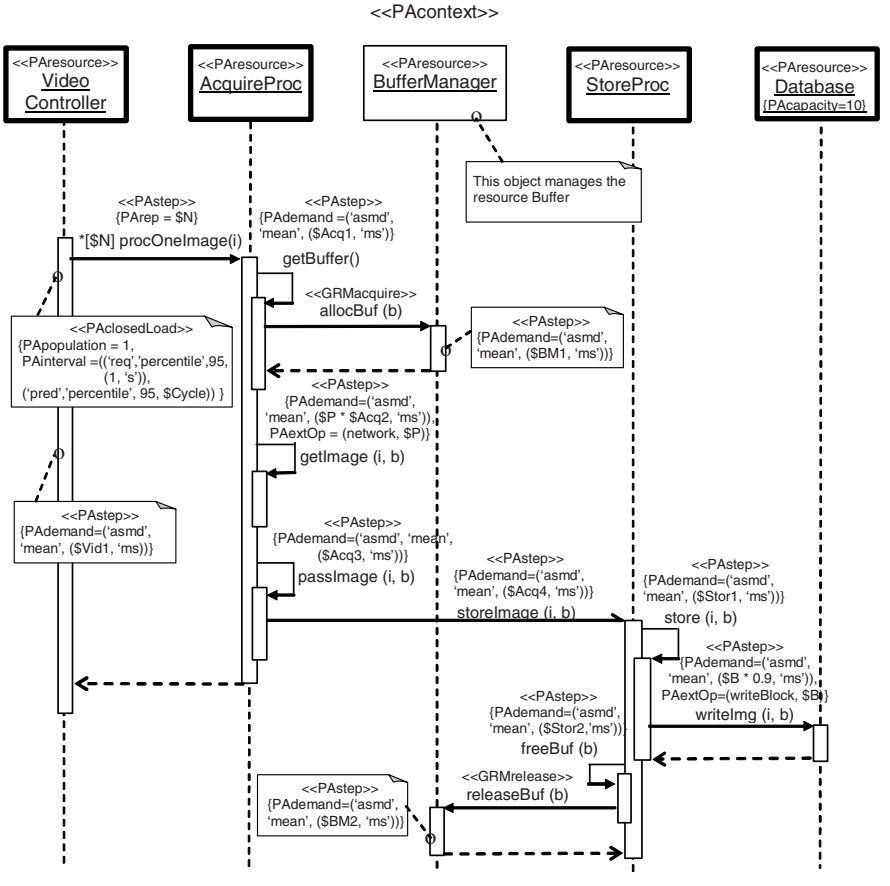


Fig. 15. Parameterized version of the video capture behaviour of Figure 14

If the differences among the instances are more severe, parameters can also be tabulated. If it is just the frame size that is different, or the number of cameras, the original Behaviour definition can be used, with parameter values for $P and $N included in the table. If every demand value is affected, every one can be replaced by a variable as illustrated in Figure 15, and they can all be tabulated.

This approach can be extended to apply to SM behaviour, and to architecture-based annotations which are applied to classes.

4.4 Parameter and Expression Syntax

The expression of performance parameters is a challenge, since all values must include the source and statistical qualifiers described in section 2.2 and it is desirable to express quantities as variables, and as expressions in other variables. In SPT this is achieved by adding qualifiers to values, and by the use of a reserved for $name for variables (as in PERL), and a simple language called TVL (tagged value language) for expressions with the basic operators +-*/.

In MARTE the qualifiers are retained and a more elaborate expression language called VSL, Value Specification Language, has been created, which allows functions, tuples, and interval values to be specified. These languages are too complex to define here.

5 Annotation of AD/SM/DD

The discussion in Section 4 has introduced the annotations for interaction diagrams (SDs), and how they relate to PM concepts. A complete description must include deployment information, which is applied to a deployment diagram (DD). Also, designers may choose to express behaviour in other ways, through activity diagrams (ADs) and state machines (SMs). To be sure of capturing the PM we must be able to annotate these also.

The reason for the different behaviour definitions comes from slightly different purposes.

- the SD expresses system traces at a certain level of abstraction. This captures the sequence of interactions, the messages between system elements. The lifelines of an SD represent system elements that interact. Using a recursive message (that activates a nested ExecutionOccurrenceSpecification, sequence internal to the element can also be expressed.
- the AD expresses causality. From [38] p 307, “Activity modeling emphasizes the sequence and conditions for coordinating lower-level behaviors, rather than which classifiers [i.e., components or subsystems] own those behaviors. These are commonly called control flow and object flow models. The actions coordinated by activity models can be initiated because other actions finish executing, because objects and data become available, or because events occur external to the flow.” The partitions of an AD may be used to designate system elements, like a lifeline of a SD, and this is assumed in SPT/MARTE. However they can also designate abstract roles: “An activity partition is a kind of activity group for identifying actions that have some characteristic in common.” [38]

p 353. Therefore attention must be paid to the significance of an activity partition, in building PM from DM.

- the SM expresses the logic of some pattern of behaviour. In SPT it was not mentioned, although SPT was applied to SMs in [5] and [7]; in MARTE SMs are included explicitly. The SM may represent the behaviour of a system element, or it may be more abstract, representing the collective state of some process.

5.1 Deployment Diagrams (DD)

The deployment diagram is required to establish the host processor for software elements, and to identify communications paths between them. In UML1 and SPT the DD shows objects allocated to nodes, as in this Figure for the buffer retrieve/store system of Figure 14:

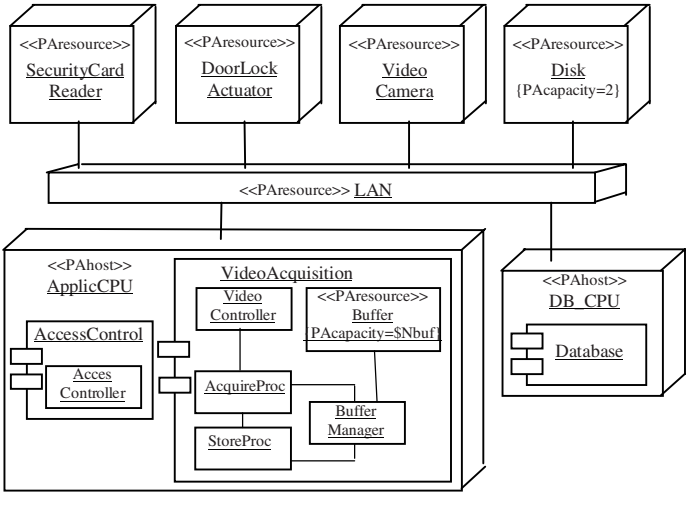


Fig. 16. Deployment for the buffer retrieve/store system of Figure 14 [25]

In this diagram the hosts are stereotyped PAhost, with a processing rate (relative to a standard host), and the deployment is indicated by components AccessControl and VideoAcquisition, with the processes inside. Only the buffer pool itself is stereotyped as a resource, since the four processes in VideoAcquisition are stereotyped in the SD of Figure 14. Note that the buffer pool resource Buffer is a separate resource from the BufferManager process... the process operates on the pool but is held only during its acquire and release operations, and not between. The pool has multiple units, defined as the variable $Nbuf, while the BufferManager is single-threaded to act as a critical section for the pool state.

The AccessControl subsystem with its peripherals SecurityCardReader and DoorLockActuator will not be discussed here.

In UML2, the semantics of the DD have changed to introduce the notion of artifacts, representing deployable files containing load modules and configuration data. They may be represented as above, or as in the following figure:

The connection between the artifact and the design objects must be established, and to simplify this MARTE provides a property of the paProcess annotation to identify the deployed artifact and thus (via the deployment association) the host.

5.2 Activity Diagrams (ADs)

The buffer system of Figure 14 is represented again in an AD as follows. Here the annotations use the MARTE NFP notation for quantities.

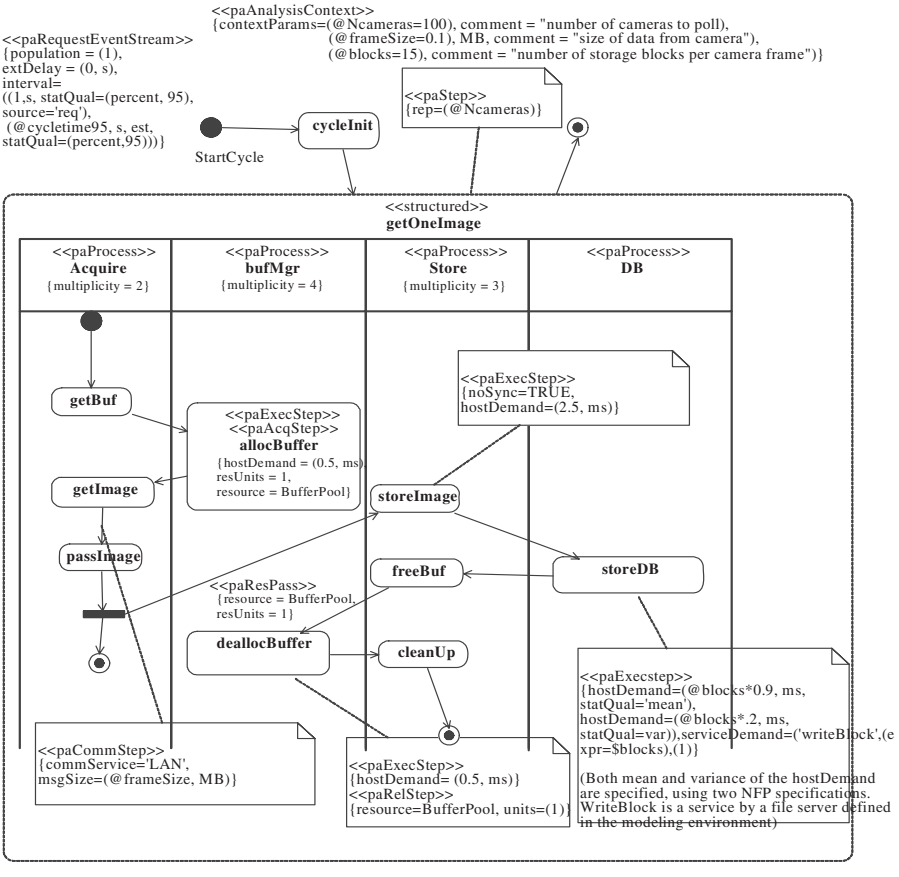


Fig. 17. AD for the buffer retrieve/store sytem

Global parameters of the analysis have been defined, using a different prefix @ suggested for MARTE (since UML now uses $ for something else). The loop over the cameras is expressed here as an explicit StructuredActivity, the large rounded rectangle with a full activity definition inside it. The swimlanes are stereotyped paProcess to identify them with deployed processes.

Notice that in AD there is also an explicit fork connector, and the same is true for join, branch and merge.

5.3 State Machine Diagrams (SMs)

State Machine diagrams define a behavior for some entity or subsystem (not necessarily for a single component or object). We need to think of three cases somewhat differently:

- a SM that responds to an input event by a self-contained behavior which terminates. This provides a *subscenario* triggered by the input event.
- a SM that responds to an input event by a self-contained behavior which runs until it waits for the next input event (the next instance of the same event). This describes a *scenario* that responds to a workload (to a stream of input events).
- a SM that runs forever once started. This spontaneously generates a workload of its own, and we call it a *WorkloadGenerator* (in MARTE).

A RequestEventStream designates requests to the system as a whole, and we assume that each system behavior is driven by one such stream only. There may be several streams and behaviours, and each one describes a user class in the performance model.

Any of these kinds of SMs may be replaced by a compound of several SMs collaborating to produce one of these three overall behaviour patterns, and communicating by signals. There are additional cases of SMs that do not respond to multiple input event streams, but they are not considered here or in the profiles.

5.3.1 Terminating SM (Sub-scenario)

The simplest SM is associated with a single component and executes a sequence of steps, then terminates. Fig 19 shows a simple web server scenario. The hostDemand requires that the allocation of the webserver component be known, to resolve the host.

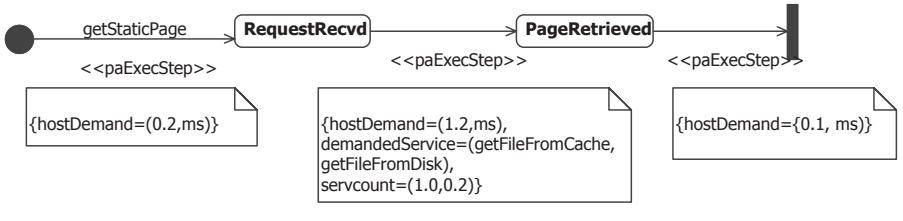


Fig. 18. A terminating SM defining a sub-scenario triggered by a request getStaticPage

The steps annotated on the transitions, as here, describe actions defined for the transitions (which can include a scenario for each one). Steps annotated on a state describe the sum of actions to be executed on arriving and leaving the state.

5.3.2 Driven SM (Scenario)

The driven SM is driven by a workload event stream (RequestEventStream), which triggers a transition from a home state such as ThreadReady to start the scenario. It is equivalent to a scenario, but all executed within the context of the SM. If the SM does not correspond to a component, then an AD may be preferable.

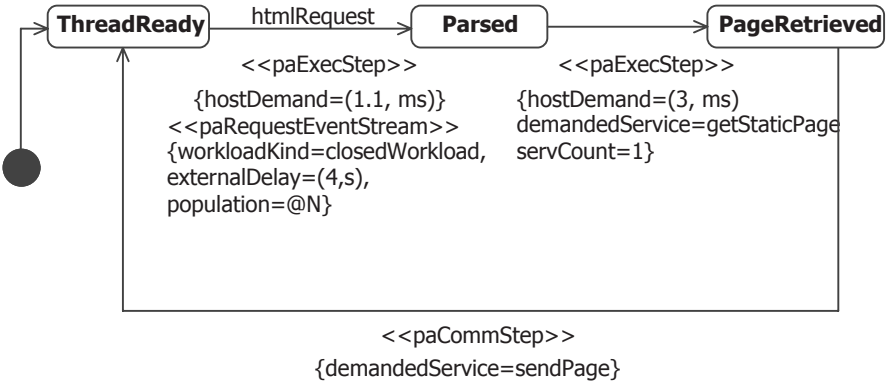


Fig. 19. A Cycling SM, defining a Scenario driven by htmlRequest

5.3.3 Perpetual SM (WorkloadGenerator)

A perpetual SM cycles forever without waiting for a trigger event; in effect it creates the trigger events in demanding subscenarios or other actions in each Step. thus it acts to generate workload for the system. The SM in Figure 21 roughly represents an abbreviated version of the sequential behaviour defined for the TPC-W benchmark, in transitioning among the pages. Each state (ExecStep) invokes a sub-scenario through a “demandedBehavior” attribute. The sub-scenario defines the nested behaviour in responding to the page, as shown for the case of GetHomePage in Figure 8. The blockingDelay for each ExecStep represents the user thinking time, which is greater for pages which require more thought. The paExecStep stereotypes applied to states define work done on entering the state; on (a few) transitions they define transition probabilities where a choice is required.

A paWorkloadGenerator stereotype applied to this diagram has an attribute “population” to define the number of concurrent users with this behavior. The population is assumed to be constant, a kind of closed workload with this defined behaviour. When one customer terminates, another is assumed to immediately begin interacting.

A corresponding open system would be modeled with this SM slightly changed, to be a sub-scenario (by having the behavior eventually terminate). The sub-scenario would be invoked by a Step in a separate behaviour triggered by an open workload.

Given the transition probabilities, a WorkloadGenerator SM becomes a Markov Chain element in the PM. Markov Chain analysis can reduce it to a probabilistic mixture of demands and delays made on the system, and thus to a set of classes of behaviour for a single customer chain in a QM or LQM analysis.

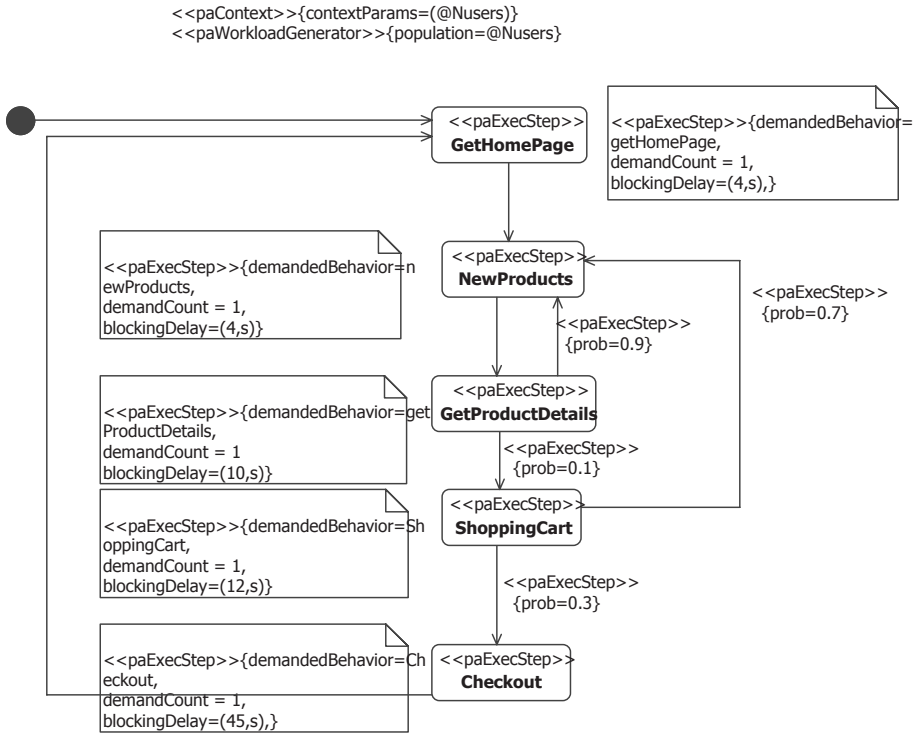


Fig. 20. A WorkloadGenerator type SM

A set of SMs may participate in terminating, cycling or perpetual behaviour by exchanging signals or messages. To interpret the behaviour as a scenario, a tool must traverse the interactions and construct the combined SM, which could be a significant tool problem.

6 Communications Modeling

It is useful and even essential to be able to define the handling of a message with more or less detail, depending on the state of the design.

- in an early stage it may be desired to ignore communications costs and delays. This is a default if no other option is defined.
- a simple way to deal with it is to attach cost parameters to the nodes, and a latency (blockingTime in MARTE) to the network. These should be sensitive to the message size. This is the default if no behaviour or service is defined.
- some protocols impose complex handshakes, authentication using servers and so on, which require a scenario to define them. If the protocol behavior is described as a scenario in the design, or in a design library that can be used, then it can be referenced directly. The scenario will have roles which need to be bound to the

sender and receiver processes, if it is to be used for many different messages. This is defined by an attribute commBehavior:paBehaviorScenario.

- the protocol may be described within a middleware layer acting as a subsystem, and its scenario can be referenced as an operation (service) of the layer. This is similar to the previous case but attaches the reference to the interface rather than to the scenario, which may exist within the middleware description. This is defined by an attribute commService: paRequestedService.
- the protocol behavior, and communications handling generally, may not be available in the UML model, but may be modeled within the PM. In this case an external operation can be used, defined by an attribute extOperation:String.

These options, apart from the first, are illustrated in Figure 22.

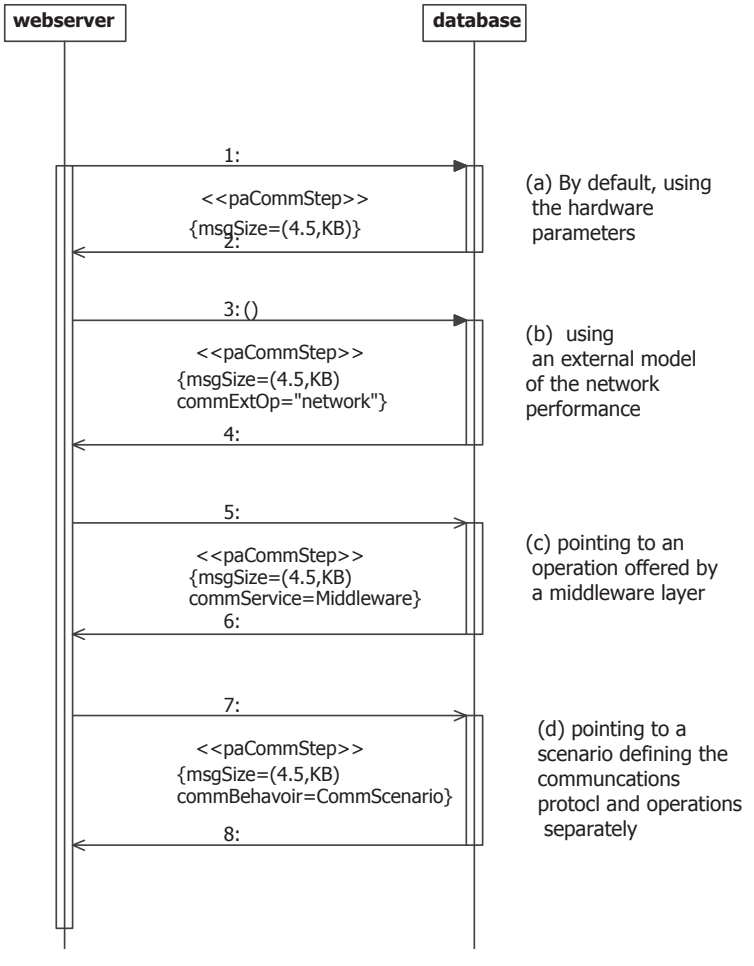


Fig. 21. Four different ways to specify the communications costs

The hook for representing communications is the CommStep, which is an annotation on a message which represents a step in the scenario, just to handle the message. The annotation may be applied to a message that is also annotated with the Step for the next operation, in which case the CommStep comes first in the scenario. Attributes of the CommStep identify which of the above options is to be applied, and identify behaviour or service operations to be included, if that is appropriate to the option. There is a constraint that only one of commService, commBehavior or extOperation may be non-null.

For the first option, the hardware communications cost parameters may be specified in the deployment diagram as in Fig 23. For the fourth option, any scenario could be used, for example the parallel read operation shown in Figure 24. This would require additional annotations to bind the database role to both of the timelines, each with separate deployed components.

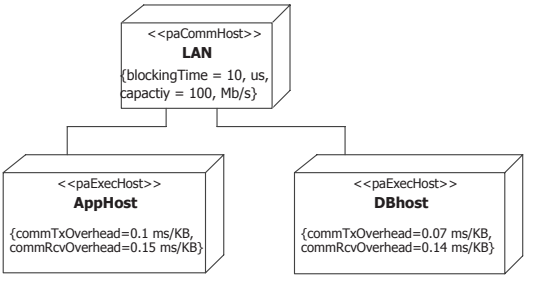


Fig. 22. Deployment with communications parameters (MARTE)

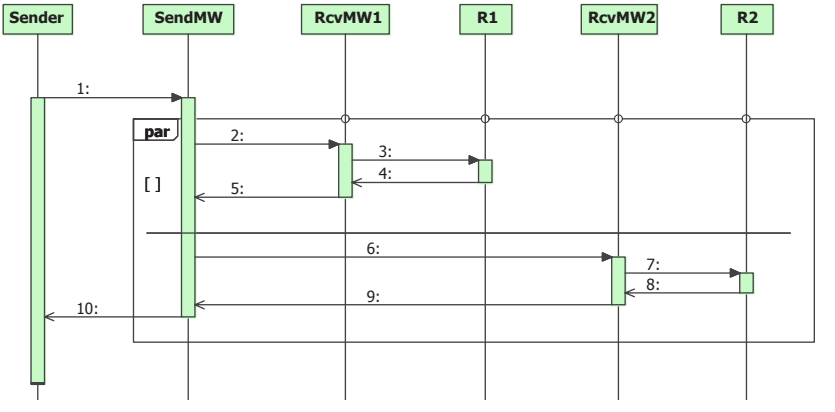


Fig. 23. A scenario showing a complex read operation involving two databases

7 Transformation Techniques

Several approaches have been taken successfully to create a performance model (PM) from a UML design model (DM). Manual creation is a solid option if automated tools

do not exist, however the DM may be constantly evolving, and a manual method can be burdensome as well as error-prone in the details (such as the copying of parameter values into the PM).

An aspect of the transformation is that the level of abstraction of the PM may be higher than that of the DM, as discussed in [44] Thus the transformation may need to aggregate concepts and quantities expressed in the DM.

7.1 Model Traversal (Ad Hoc Construction)

To construct the performance model, the commonest technique has been to traverse in some ad-hoc fashion the UML model represented in the form of a DOM-tree (domain object model tree) which is derived from the XML format output by most tools (called XMI). During the traversal, objects in the PM domain are constructed with hints to their relationships to other objects. Finally the PM objects are assembled. If the PM is a Petri net or stochastic process algebra model the PM structure is close enough to the UML to allow direct construction of the PM by this approach.

7.2 Use of an Intermediate Model

Some approaches have proceeded by way of an intermediate model between the DM and the PM. Examples are

- the CSM (Core Scenario Model) described earlier [40]. This intermediate model was partly to assist the transformation and partly to generalize it.
- KLAPER [9], which supports a broad range of modeling domains.
- the Execution Graph used by Cortellessa and Mirandola in [8].

For example in PUMA [24] the Core Scenario Model (CSM) has three roles, to filter the relevant information from the DM, to support a PM structure which is quite different from the DM structure, and to unify information from different styles of behavior definition. Then the PM is constructed by traversal of the DOMtree of the CSM.

7.3 XSLT

A general tool for transforming XML languages is XSLT [48]. This is intended to transform model elements in one language into corresponding elements in another, for example for alternative rendering of graphical images, or for translating design behavior into code. If the PM is sufficiently close to the UML in structure (essentially, for network-type models) this is practical, in other cases it is less so. Smith and Llado have used it to translate Execution Graphs to a QN description in XML called PMIF [46]. To generate LQN models however it was necessary to construct entities that do not exist in the DM, and an ad hoc transformation was used instead.

7.4 Graph Grammar

The problem of inferring structures in the DM behavior can be attacked by pattern matching, and the class of graph grammars provide a more general approach than say

XSLT. They were used in [19] to create LQMs. The DM is first interpreted as a graph abstraction, which is a kind of intermediate model, then the grammar is used to define transformations to an output graph representing the model.

7.5 QVT

The QVT (Query/View/Transform) standard [37] was initiated to support transforming UML models to achieve model-driven development, but it is more general. It transforms from a source model based on one MOF metamodel, to a target model based on the same or another MOF metamodel. A running example in [37] is to create a database schema from a UML data model.

The approach of QVT is express conditions which express patterns in the source model, and when the conditions are satisfied, to take actions on the elements of the matched pattern to create the target model. Grassi et al used QVT for successive transformations to create models for component-based systems in [10].

8 Conclusion

This conceptual overview addresses the semantic relationships between design model annotations, and performance models. The concept of “completions” which are needed to make the evaluation feasible, is central. Completions include the annotations themselves, and platform and environment representation either in the DM, the annotations, or in performance model elements outside the DM. This adaptability is necessary, to adapt to different levels of completion of the DM.

The reader is reminded again that the annotations presented in the examples come from multiple versions of the performance profile, and they should ignore the inconsistencies in syntax and look at the underlying information. For precise syntax they should consult the SPT standard, or the MARTE proposal expected shortly.

References

To identify the literature of DM-to-PM translation, it is grouped together in a separate subsection. As a bibliography however it does not pretend to be complete.

Performance Models from Annotated Design Models

[1] A. Alsaadi, "A Performance Analysis Approach based on the UML Class Diagram," in *Proc. 4th Int. Conf. on Software and Performance (WOSP 2004),* Redwood City, CA, Jan 2004, pp. 254-260.

[2] L. B. Arief and N. A. Speirs, "A UML Tool for an Automatic Generation of Simulation Programs," in Proceedings of the Second International Workshop on Software and Performance (WOSP2000), Ottawa, Canada, September 17-20, 2000, pp. 71-76.

[3] S. Balsamo, M. Marzolla, “Performance evaluation of UML software architectures with multiclass Queueing Network models”, Proc.5th Int. workshop on Software and performance, Palma, Pages: 37 – 42, 2005

[4] A. J. Bennett, A. J. Field, and C. M. Woodside, "Experimental Evaluation of the UML Profile for Schedulabiity, Performance and Time," in Proc. UML 2004, v. 3273 of Lecture Notes in Computer Science (LNCS 3273), Lisbon, Oct 2004, pp. 143-157.

[5] S. Bernardi, S. Donatelli, and J. Merseguer, "From UML sequence diagrams and statecharts to analysable Petri net models," in *Proc. 3rd Int. Workshop on Software and Performance*, Rome, July 2002, pp. 35-45.

[6] M. Bozga, S. Graf, L. Mounier, I. Ober, J.-L. Roux, and D. Vincent, "Timed Extensions for SDL," in Proc. SDL Forum 01, Copenhagen, June, 2001.

[7] C. Canevet, S. Gilmore, J. Hillston, M. Prowse, and P. Stevens. Performance modelling with UML and stochastic process algebras. IEE Proceedings: Computers and Digital Techniques, 150(2):107-120, March 2003

[8] V. Cortellessa and R. Mirandola, "Deriving a Queueing Network based Performance Model from UML Diagrams," in *Proc. Second Int. Workshop on Software and Performance,* Ottawa, September 2000, pp. 58-70.

[9] V. Cortellessa, A.D'Ambrogio, and G. Iazeolla, "Automatic derivation of software performance models from CASE documents," Performance Evaluation, vol. 45, pp. 81-105, 2001.

[10] V. Grassi, R. Mirandola, A. Sabetta, "Filling the gap between design and performance/reliability models of component-based systems: A model-driven approach", Journal of Systems and Software v. 80, pp 528–558, 2007.

[11] J. Hillston and Y. Wang. Performance evaluation of UML models via automatically generated simulation models. In S. A. Jarvis, editor, UK Performance Engineering Workshop, pages 64–78, July 2003.

[12] P. Kahkipuro, "UML Based Performance Modeling Framework for Object Oriented Systems," in *UML99, The Unified Modeling Language, Beyond the Standard, LNCS 1723,* Springer-Verlag, Berlin, 1999, pp. 356-371.

[13] J. P. Lo'pez-Grao, J. Merseguer, and J. Campos, "From UML Activity Diagrams To Stochastic Petri Nets" in *Fourth Int. Workshop on Software and Performance*, Redwood City, CA, Jan. 2004, pp. 25-36.

[14] J. Merseguer, *Software performance engineering based on UML and Petri nets*, Ph.D. thesis, University of Zaragoza, Spain, March 2003.

[15] J. Merseguer, J. Campos, and E. Mena, "A Pattern-Based Approach to Model Software Performance," in Proc.2nd Int. Workshop on Software and Performance (WOSP2000), Ottawa, 2000, pp. 137-142.

[16] A. Mitschele-Theil and B. Muller-Clostermann, "Performance Engineering of SDL/MSC Systems," *Journal on Computer Networks and ISDN Systems*, vol. 31, no. 17 pp. 1801-1815, 1999.

[17] D.B. Petriu and M. Woodside, "Software Performance Models from System Scenarios in Use Case Maps", in *Proc. 12th Int. Conf. on Modelling Tools and Techniques*, London, England, April 2002.

[18] D. B. Petriu, D. Amyot, and C. M. Woodside, "Scenario-Based Performance Engineering with UCMNav," in Proc 11th Int. SDL Forum (SDL 2003), Springer, LNCS v 2708, 2003, pp. 18 - 35.

[19] D. C. Petriu and H. Shen, "Applying the UML Performance Profile: Graph Grammar-based derivation of LQN models from UML specifications," in *Proc. 12th Int. Conf. on Modelling Tools and Techniques for Computer and Communication System Performance Evaluation,* London, England, 2002.

[20] D. C. Petriu and C. M. Woodside, "Performance Analysis with UML," in *UML for Real*., B. Selic, L. Lavagno, and G. Martin, Eds. Kluwer, 2003, pp. 221-240.

[21] R. Pooley, "Software Engineering and Performance: a Roadmap," in The Future of Software Engineering, part of the 22nd Int. Conf. on Software Engineering (ICSE2000), Limerick, Ireland, June 2000, pp. 189-200.
[22] L. Pustina, S. Schwarzer, M. Gerharz, P. Martini, V. Deichmann, "Performance Evaluation of a DVBH Enabled Mobile Device System Model", Proc 6th Int. Workshop on Software and Performance (WOSP 2006), Buenos Aires, Feb. 2007, pp 164-171.
[23] A. Schmietendorf and E. Dimitrov, "Possibilities of Performance Modeling with UML," in *Performance Engineering*, R. Dumke, C. Rautenstrauch, A. Schmietendorf, and A. Scholz, Eds. Springer-Verlag, 2002, pp. 78-95.
[24] M. Woodside, D.C. Petriu, D.B. Petriu, H. Shen, T. Israr, J. Merseguer, "Performance by Unified Model Analysis (PUMA)", *Proc. 5th Int. Workshop on Software and Performance* , Palma de Mallorca, July 2005, pp 1-12.
[25] J. Xu, M. Woodside, and D.C. Petriu, "Performance Analysis of a Software Design using the UML Profile for Schedulability, Performance and Time," in *Proc. 13th Int. Conf. on Modelling Techniques and Tools for Computer Performance Evaluation*, Urbana, USA, Sept. 2003.

Other References

[26] S. Balsamo and M. Marzolla. "Simulation Modeling of UML Software Architectures", *Proc. ESM'03,* Nottingham (UK), June 2003
[27] G. Bolch, S. Greiner, H. De Meer, K. S. Trivedi, *Queueing Networks and Markov Chains*, Wiley-Interscience, 1998.
[28] C. Cavenet, S. Gilmore, J. Hillston, L. Kloul, and P. Stevens, "Analysing UML 2.0 activity diagrams in the software performance engineering process," in *Proc. 4th Int. Workshop on Software and Performance*, Redwood City, CA, Jan 2004, pp. 74-83.
[29] Espinoza, H., Dubois, H., Gerard, S., Medina, J., Petriu, D.C. and Woodside M., "Annotating UML Models with Non-Functional Properties for Quantitative Analysis," in MoDELS 2005 Workshops (Jean-Michel Bruel, Ed.), LNCS 3844, pp. 79-90, Springer-Verlag, 2006
[30] G. Franks, S. Majumdar, J. Neilson, D. Petriu, J. Rolia, and M. Woodside, *Performance Analysis of Distributed Server Systems*, Proc. Sixth International Conference on Software Quality, Ottawa, Canada, 1996, pp. 15-26.
[31] G. Franks, "Performance Analysis of Distributed Server Systems", PhD. thesis, Carleton Uni-versity, Jan. 2000.
[32] G. Franks, P. Maly, M. Woodside, D. C. Petriu, and A. Hubbard, "Layered Queueing Network Solver and Simulator User Manual," Dept. of Systems and Computer Engineering, Carleton University, Ottawa., Dec. 2005, at http://www.sce.carleton.ca/rads/lqn/lqn-documentation/
[33] R. Jain, The Art of Computer Systems Performance Analysis. John Wiley & Sons Inc., 1991
[34] P. Maly and C. M. Woodside, "Layered Modeling of Hardware and Software, with Application to a LAN Extension Router", *Proc 11th Int Conf on Computer Performance Evaluation Techniques and Tools* (TOOLS 2000), Chicago, Mar. 2000, pp. 10 - 24.
[35] Object Management Group, *UML Profile for Schedulability, Performance, and Time Specification*, OMG Adopted Specification ptc/02-03-02, July 1, 2002.
[36] Object Management Group, "UML Profile for Modeling and Analysis of Real-Time and Embedded systems (MARTE) RFP", document realtime/05-02-06, Feb. 6, 2005.

[37] Object Management Group, "MOF QVT Final Adopted Specification," ptc/05-11-01, Nov. 2005.
[38] Object Management Group, "Unified Modeling Language: Superstructure, version 2.1," ptc/2006-04-02, 2006.
[39] D. B. Petriu and M. Woodside, "Analysing Software Requirements Specifications for Performance," in Third Int. Workshop on Software and Performance (WOSP 02), Rome, Italy, July 2002, pp. 1 - 9.
[40] D. B. Petriu and M. Woodside, "An intermediate metamodel with scenarios and resources for generating performance models from UML designs", Software and Systems Modeling, vol. 5, no. 4 2006.
[41] D. Pilone and N. Pitman, UML 2.0 in a Nutshell. O'Reilly, 2005.
[42] F. Sheikh and C. M. Woodside, "Layered Analytic Performance Modelling of Distributed Database Systems", *Proc. Int. Conf. on Distributed Computer Systers*, Baltimore, U.S.A., May 1997, pp. 482-490.
[43] J. R. Rolia and Kenneth Sevcik, "The method of layers", *IEEE Transactions on Software Engineering*, Vol. 21, No. 8, 1995, pp. 689-700
[44] A. Sabetta, D. C. Petriu, V. Grassi, and R. Mirandola, "Abstraction-raising Transformation for Generating Analysis Models," in Proc. of MoDELS'2005 Satellite Events: Revised Selected Papers. vol. 3844, LNCS, J. M. Bruel, Ed. Springer, 2006, pp. 217-226.
[45] C. U. Smith and L. G. Williams, *Performance Solutions*. Addison-Wesley, 2002.
[46] C. U. Smith and C. M. Llado, "Performance Model Interchange Format (PMIF 2.0): XML Definition and Implementation," in First Int. Conf. on Quantitative Evaluation of Systems (QEST '04), Enschede, 2004, pp. 38-47.
[47] C. U. Smith, C. M. Lladó, V. Cortellessa, A. D. Marco, and L. G. Williams, "From UML models to software performance results: an SPE process based on XML interchange formats," in Proc 5th int workshop on Software and performance, Palma de Mallorca, 2005, pp. 87-98.
[48] D. Tidwell, XSLT. O'Reilly, 2001.
[49] Transaction Processing Council, "TPC Benchmark W (Web Commerce) Specification", Version 1.8, Feb 19, 2002
[50] T. Verdickt, B. Dhoedt, F. D. Turck, and P. Demeester, "Hybrid Performance Modeling Approach for Network Intensive Distributed Software," in Proc. 5th Int. Workshop on Software and Performance (WOSP 07), Buenos Aires, Feb 2007
[51] M. Woodside, C. Hrischuk, B. Selic, and S. Bayarov, "Automated Performance Modeling of Software Generated by a Design Environment," *Performance Evaluation*, vol. 45, no. 2-3 pp. 107-124, 2001.
[52] C. M. Woodside, "Software Resource Architecture", *Int. Journal on Software Engineering and Knowledge Engineering (IJSEKE)*, vol. 11, no. 4 pp. 407-429, 2001.
[53] M. Woodside, DB. Petriu, K.H. Siddiqui, "Performance-related Completions for Software Specifications", Proc 24th Int. Conf. on Software Engineering, May 2002.

Appendix: Compare SPT and MARTE

This comparison may help to keep straight some of the examples which use one notation or the other. It is very brief and only hits the high spots of the corresponding features of both the existing profile and the one which is being formulated. There is unfortunately no published version of MARTE at the time of writing.

SPT [35]	MARTE
PanalysisContext	paAnalsyisContext [adds parameters]
Pscenario	paBehaviorScenario
Pstep, attribute hostDemand	paStep [for pure behavior and sub-scenarios]
	attribute noSync
	paExecStep [primitive step]
	attributes requestedService, servCount
	attributes demandedBehavior, behavCount
	attributes extOperation, extOpCount
	paCommStep
	attributes commService, commBehavior, extOperation
	gaAcqStep, attributes Resource, resUnits
	gaRelStep, attributes Resource, resUnits
	paResPassStep, attributes Resource, resUnits
PclosedLoad	paRequestEventStream {workloadKind = closed}
attributes population, externalDelay	similar
PopenLoad	paRequestEventStream {workloadKind = open}
attribute rate	similar
	paRequestedService, applied to *Operation*, inherits ExecStep attributes
Presource	gaResource [abstract resource concept]
	paProcess [process thread pool]
	gaCommChannel [layer subsystem]
	paLogicalResource [other logical resources]
Phost	paExecHost [processor]
	paCommHost [network, bus, link hardware]

Author Index

Balbo, Gianfranco 83
Balsamo, Simonetta 34
Bernardo, Marco 180
Bradley, Jeremy T. 318

Ciardo, Gianfranco 371
Clark, Allan 132

Gilmore, Stephen 132
Gribaudo, Marco 271

Hillston, Jane 132

Knottenbelt, William J. 318
Kwiatkowska, Marta 220

Marin, Andrea 34

Norman, Gethin 220

Parker, David 220

Smith, Connie U. 395
Stewart, William J. 1

Telek, Miklós 271
Tribastone, Mirco 132

Woodside, Murray 429