



Università  
Ca' Foscari  
Venezia

**Scuola Dottorale di Ateneo  
Graduate School**

**Dottorato di ricerca  
in Informatica  
Ciclo XXVI  
Anno di discussione 2013**

***On the Solution of Cooperating Stochastic Models***

**SETTORE SCIENTIFICO DISCIPLINARE DI AFFERENZA: INF/01  
Tesi di Dottorato di Gian-Luca Dei Rossi, matricola 955834**

**Coordinatore del Dottorato**

**Prof. Riccardo Focardi**

**Tutore del Dottorando**

**Prof.ssa Simonetta Balsamo**



*To the future readers this thesis, for their patience.*



# Abstract

Stochastic models are widely used in the performance evaluation community. In particular, Markov processes, and more precisely, Continuous Time Markov Chains (CTMCs), often serve as underlying stochastic processes for models written in higher-level formalisms, such as Queueing Networks, Stochastic Petri Nets and Stochastic Process Algebras. While compositionality, i.e., the ability to express a complex model as a combination of simpler components, is a key feature of most of those formalisms, CTMCs, by themselves, don't allow for mechanisms to express the interaction with other CTMCs. In order to mitigate this problem various lower-level formalisms have been proposed in literature, e.g., Stochastic Automata Networks (SANs) [146], Communicating Markov Processes [46], Interactive Markov chains [100] and the labelled transition systems derived from PEPA models [101].

However, while the compositionality of those formalism is a useful property which makes the modelling phase easier, exploiting it to get solutions more efficiently is a non-trivial task. Ideally one should be able to either detect a product-form solution and analyse the components in isolation or, if a product form cannot be detected, use other techniques to reduce the complexity of the solution, e.g., reducing the state space of either the single components or the joint process. Both tasks raised considerable interest in the literature, e.g., the RCAT theorem [89] for the product-form detection or the Strong Equivalence relation of PEPA [101] to aggregate states in a component-wise fashion.

This thesis deals with the aforementioned problem of efficiently solving complex Markovian models expressed in term of multiple components. We restrict our analysis to models in which components interact using an active-passive semantics. The main contributions rely on automatic product-forms detection [15, 9, 22], in components-wise lumping of forward and reversed processes [12, 11] and in showing that those two problems are indeed related, introducing the concept of *conditional product-forms* [7].

*Structure of the thesis* This work is divided in three parts. The first one gives to the reader a general introduction to stochastic modelling, with a particular focus on Markovian models, and to the formalisms that will appear throughout the thesis. Moreover it gives some basic notions about product-form solutions and an overview of available tools for multiformalism modelling, which is needed in order to understand some of the applications that appear later in the thesis.

Part 2 deals with the main contributions of the thesis. First, it introduces algorithmic product form detection and solution techniques, and a tool for designing, detecting and solving product-form stochastic models in a compositional and modular way. It then presents a new criterion for component-wise state space reduction, in a way similar to PEPA's strong equivalence. Finally, it introduces the concept of *conditional product form*, and it shows the relation between this notion and the lumpability, according to our criterion, of reversed processes, thus linking the two main topics of this work. This research has both a theoretical significance and a practical impact, since all the aforementioned contributions allows for the efficient solution of stochastic models for which an analysis was previously unfeasible.

Part 3 shows some applications of the aforementioned techniques to the analysis of complex models, with a particular emphasis on heterogeneous models, i.e., models whose components exhibit different behaviours and can even be expressed using different higher level formalisms. An application of product-form theory to some classes of stochastic Petri nets is also given.

The conclusion recapitulates the results of the thesis and analyses the impact of the work on the performance evaluation community. Finally, a forecast on possible future developments is discussed.

# Acknowledgments

This thesis contains part of the work that I did during my Ph.D. studies. This activity has led to a number of publications. For all the those papers I have given an original and concrete contribution, however none of them would be born without my co-authors. In particular I want to mention the fundamental help, guidance and inspiration that I have been given by my supervisor, Simonetta Balsamo, and by Andrea Marin, with whom I did most of my day-to-day work. I want also to thank Lucia Gallina and Sabina Rossi for having made me aware of formal verification and model checking issues in performance evaluation. Thanks to William J. Knottenbelt and Enrico Vicario for their reviews of the preliminary version of this thesis, which would have been much less refined in its content and form without their suggestions.

I finally have to remember all the other colleagues, co-authors, friends and family members that have helped me during those 3 years full of events. I cannot name them one by one without the risk to forget somebody.





---

# Contents

Preface	ix
Introduction	xi
<b>I An introduction to stochastic modelling</b>	<b>1</b>
<b>1 Stochastic models</b>	<b>3</b>
1.1 Introduction . . . . .	3
1.2 Markovian models . . . . .	3
<b>2 Formalisms and cooperation semantics</b>	<b>7</b>
2.1 Queueing Networks . . . . .	7
2.2 Stochastic Petri nets . . . . .	8
2.2.1 Stochastic Petri nets . . . . .	9
2.2.2 Building blocks . . . . .	10
2.3 Markovian Process Algebra . . . . .	11
2.3.1 Classical process algebras . . . . .	11
2.3.2 Process algebras extensions . . . . .	13
2.3.3 Performance Evaluation Process Algebra . . . . .	14
2.4 Stochastic Automata . . . . .	16
<b>3 Product Forms</b>	<b>19</b>
<b>4 Tools</b>	<b>23</b>
4.1 Introduction . . . . .	23
4.2 Software Analysis . . . . .	26
4.3 A model example . . . . .	33
4.4 Software Comparison . . . . .	35
4.5 Conclusion . . . . .	37
<b>II Contributions</b>	<b>41</b>
<b>5 Algorithmic Product form detection and solution</b>	<b>43</b>
5.1 Introduction . . . . .	43
5.2 The INAP Algorithm . . . . .	44
5.3 INAP for models with infinite state spaces . . . . .	46

5.3.1	The algorithm input . . . . .	46
5.3.2	Main idea of the algorithm . . . . .	47
5.3.3	Formal definition of INAP+ . . . . .	48
5.3.4	Convergence, termination, complexity and optimisations . . . . .	50
5.4	Model and cooperation encoding . . . . .	50
5.5	Tool . . . . .	52
5.5.1	Specifying the interactions . . . . .	53
5.5.2	Client-server architecture . . . . .	54
5.5.3	Use cases . . . . .	55
5.5.4	MSI implementation example . . . . .	56
5.6	A Numerical Example . . . . .	57
5.7	Conclusions . . . . .	60
<b>6</b>	<b>Component-wise state space reduction</b>	<b>63</b>
6.1	Introduction . . . . .	63
6.1.1	Related work . . . . .	63
6.1.2	Contribution . . . . .	65
6.2	Exact lumpability . . . . .	65
6.2.1	Exact lumping and strong equivalence . . . . .	70
6.3	Conclusions . . . . .	72
<b>7</b>	<b>Conditional Product-Forms</b>	<b>73</b>
7.1	Introduction . . . . .	73
7.1.1	Feed-forward synchronisations . . . . .	73
7.2	Conditional product-form and lumping of the reversed automata . . . . .	75
7.2.1	Theoretical considerations about Theorem 3 and 4 . . . . .	80
7.3	Conclusions . . . . .	84
<b>8</b>	<b>Approximate aggregation techniques</b>	<b>85</b>
8.1	Introduction . . . . .	85
8.2	Evaluation of the quality of clusters . . . . .	85
8.3	Algorithm definition . . . . .	86
8.4	Example . . . . .	89
8.5	Conclusions . . . . .	92
<b>III</b>	<b>Case studies and applications</b>	<b>93</b>
<b>9</b>	<b>Exploiting product forms solution techniques in multiformalism modelling</b>	<b>95</b>
9.1	Introduction . . . . .	95
9.2	From multiformalism models to product-form solutions . . . . .	96
9.2.1	Deciding and computing the product-form solution . . . . .	97

---

9.2.2	The formalisms . . . . .	99
9.3	Case study . . . . .	101
9.3.1	Overall model description . . . . .	102
9.3.2	Model specification . . . . .	103
9.3.3	Model analysis and results . . . . .	105
9.4	Conclusions and future work . . . . .	109
<b>10</b>	<b>Modelling retrial-upon-conflict systems with product-form stochastic Petri nets</b>	<b>111</b>
10.1	Introduction . . . . .	111
10.2	The conflict model . . . . .	112
10.3	Applications . . . . .	118
10.3.1	A computer network with collisions . . . . .	118
10.3.2	A transactional database system . . . . .	122
10.4	Conclusions . . . . .	125
	<b>Conclusions</b>	<b>127</b>
C.1	Contributions . . . . .	127
C.2	Impact and Future Works . . . . .	127
	<b>Bibliography</b>	<b>129</b>



---

# List of Figures

2.1	Tandem of exponential queues. . . . .	17
4.1	SHARPE: a Multiclass Product Form Queueing Network . . . . .	27
4.2	SHARPE: composed model, RBD . . . . .	28
4.3	SHARPE composed model, Markov chain . . . . .	28
4.4	Plotting capabilities of SHARPE . . . . .	29
4.5	Möbius: a composed model . . . . .	30
4.6	Möbius: a PEPA <sub>k</sub> submodel . . . . .	31
4.7	Möbius: a SAN submodel . . . . .	31
4.8	Möbius: simulation results . . . . .	32
5.1	CTMC underlying a G-queue with catastrophes. . . . .	45
5.2	Example of CTMC underlying a Jackson queueing station. . . . .	46
5.3	Tandem of two exponential finite capacity queues. . . . .	52
5.4	Example of Figure 5.3 model with varied probabilistic routing. . . . .	53
5.5	Types of connections between labels. . . . .	53
5.6	Truncation of the birth and death process underlying an exponential queue. . . . .	56
5.7	Jackson network of Example 7. . . . .	56
5.8	Screenshot of the model corresponding to the Jackson network of Figure 5.7. . . . .	57
5.9	Screenshot of the window for the synchronisation details. . . . .	57
5.10	Example of a heterogeneous queueing network. . . . .	58
6.1	Example of automaton with types $T = \{1, 2, 3, 4\}$ . Arcs are labelled by their type $\ell$ and the rate $\lambda_\ell \mathbf{E}_{1\ell}(s_1, s'_1)$ . Self-loops associated with type 2 are omitted for the sake of clarity. . . . .	71
6.2	Lumping of the automaton of Figure 6.1. Arcs are labelled by their type $\ell$ and the rate $\tilde{\lambda}_\ell \tilde{\mathbf{E}}_{1\ell}$ . Self-loops associated with type 2 are omitted for the sake of clarity. . . . .	71
7.1	Graphical representation of a G-network and the corresponding model using automata. . . . .	83
7.2	Exponential queues with synchronised arrivals and their representation by stochastic automata. . . . .	83
8.1	Example model. . . . .	90
8.2	Automata of the example model. . . . .	91

---

9.1	Formalisms elements . . . . .	100
9.2	Overall model description . . . . .	102
9.3	Submodel QN1 . . . . .	103
9.4	Submodel QN2 . . . . .	104
9.5	SPN submodels . . . . .	106
9.6	Submodels composition . . . . .	107
9.7	Labelled Exponential Automata produced by SIMTHESysER . . . . .	108
10.1	The main building block with 2 places . . . . .	113
10.2	The main building block with 3 places . . . . .	114
10.3	The conflict building block . . . . .	115
10.4	A complete model for $l = 2$ . . . . .	116
10.5	Average response time as a function of packet arrival rate, with different number of stations . . . . .	120
10.6	Average response time as a function of the number of stations . . . . .	121
10.7	Average response time as a function of the arrival rate of transactions, with different number of processors . . . . .	123
10.8	Maximum arrival rate to each processor as a function of the number of processors, with different conflict probabilities . . . . .	124

---

# List of Tables

4.1	Characteristics of multi-formalism software packages. . . . .	38
5.1	Parameters . . . . .	61
5.2	$\rho$ and $n$ . . . . .	61
5.3	Results . . . . .	61
7.1	Rates of the reversed automaton corresponding to the automaton of Figure 6.1. . . . .	81
8.1	Ideal algorithm for computing the approximated marginal distribu- tions of cooperating automata. . . . .	87
8.2	Comparison between approximation methods. . . . .	92
10.1	Parameter values for the example of Section 10.3.1 . . . . .	119
10.2	Parameter values for the example of Section 10.3.2 . . . . .	123





---

# Preface

In this thesis we present some of the results that we obtained during the 3 years of my Ph.D. (starting from September 2010), and that concern, as the title states, the solution of stochastic models that cooperate among each other. The relevance of those kind of models has been widely stressed in literature, with a particular focus on the efficiency of their solutions. Our work in this field led to some contributions to the state of the art in the fields of product-form detection and solution [15, 9] and component-wise states space reduction [12, 11]. We have then shown that those two topics are indeed strongly related through the introduction of *conditional product-forms* [7]. Moreover, in this thesis we consider some of the applications of the aforementioned results to the solution of models expressed in multiple formalisms [13, 22] and of particular classes of stochastic Petri nets [14].

While the aforementioned issues have been core topics of my Ph.D., I also worked in several other areas that are not mentioned in this thesis, such as the translation of models expressed in a high-level hierarchical formalism to BCMP networks [8], the use of matrix-geometric methods to optimise garbage collectors' performances [10] and packets fragmentation in wireless networks [66], the use of semi-Markov models and simulation to devise a heuristic for energy consumption optimisation in ARQ protocols [67], the use of formal methods and process algebras to evaluate the resistance of mobile networks to jamming [77] and of probabilistic model checking to evaluate the performances of cognitive wireless networks and formally verify some of their properties [65].

For all the papers that I cited in this preface I have given an original and concrete contribution, however none of them would be born without my co-authors. In particular I want to mention the fundamental help, guidance and inspiration that I have been given by my supervisor, Simonetta Balsamo, and by Andrea Marin, with whom I did most of my day-to-day work. I want also to thank Lucia Gallina and Sabina Rossi for having made me aware of formal verification and model checking issues in performance evaluation.



---

# Introduction

Stochastic models are widely used in the performance evaluation community. In particular, Markov processes, and more precisely, Continuous Time Markov Chains (CTMCs), often serve as underlying stochastic processes for models written in higher-level formalisms, such as Queueing Networks, Stochastic Petri Nets and Stochastic Process Algebras. While compositionality, i.e., the ability to express a complex model as a combination of simpler components, is a key feature of most of those formalisms, CTMCs, by themselves, don't allow for mechanisms to express the interaction with other CTMCs. In order to mitigate this problem various lower-level formalisms have been proposed in literature, e.g., Stochastic Automata Networks (SANs) [146], Communicating Markov Processes [46], Interactive Markov chains [100] and the labelled transition systems derived from PEPA models [101].

However, while the compositionality of those formalism is a useful property which makes the modelling phase easier, exploiting it to get solutions more efficiently is a non-trivial task. Ideally one should be able to either detect a product-form solution and analyse the components in isolation or, if a product form cannot be detected, use other techniques to reduce the complexity of the solution, e.g., reducing the state space of either the single components or the joint process. Both tasks raised considerable interest in the literature, e.g., the RCAT theorem [89] for the product-form detection or the Strong Equivalence relation of PEPA [101] to aggregate states in a component-wise fashion.

## Prerequisites

In order to read this thesis the reader should be familiar with the basic notions of probability theory, and in particular with the negative exponential distribution. For a quick review on these topics, see Chapters 1-8 of [165]. In Part I, some background notions and some informations on the state of the art are given. Chapter 1 gives an introduction to stochastic models, with a particular focus on Markov Chains. Chapter 2 introduces some of the basic notions about some of the most well known higher level formalisms for stochastic modelling, as well as the semantics of the interaction among components expressed using the same formalism. We consider, in particular, queueing networks, stochastic Petri nets, stochastic process algebras and stochastic automata. In Chapter 3 we give a very informal introduction to product form decomposition and to the most important developments of its theory in the last decade. Finally, in Chapter 4 we survey the most important tools for the solution of stochastic models, with a particular focus on those that support the interaction of components expressed using different formalisms.

## Main contributions

Part II deals with the main results we obtained on the solution of cooperating stochastic models. The main contributions of the thesis rely on automatic product-forms detection (Chapter 5), in components-wise lumping of forward and reversed processes (Chapter 6) and in showing that those two problems are indeed related, introducing the concept of *conditional product-forms* (Chapter 7). We finally propose an approximation technique for models that cannot be exactly aggregated (Chapter 8).

The results illustrated in this thesis have both a theoretical significance and a practical impact, since the reduction of the solutions' spatial and computational complexity allows for the development of tools that are capable of efficiently solving models that are intractable with current softwares and methodologies due to the cardinality of their states space.

Chapter 5 illustrates a tool, initially presented in [15], that given the description of a set of cooperating CTMCs (i.e., when some transitions in one chain force transitions in other chains) it decides whether the model is in product-form and, in this case, computes its stationary distribution. The tool is based on the algorithm presented in [132]. Since the analysis of the product-form is performed at the CTMC level, it is able to study product-form models that are originated from different formalisms, such as exponential queueing networks, G-networks or queueing networks with blocking. To this aim, we observe that it is important to decouple the analyser and the model specification interface (MSI). We propose both a Java implementation of the analyser and a general MSI. Note that multiple specification interfaces may be implemented according to the modeller needs. With this tool, a modeller has a library of product-form models that, even if they were created using some (possibly high-level) formalism, are stored as stochastic automata, basically a CTMC with labelled transitions allowing self-loops or multiple arcs between states. Using the MSI, which acts as a client with respect to the analyser, the various sub-models can be instantiated and their interactions be specified. The operations that the modeller performs in the MSI are translated into commands for the server side, i.e., the analyser. The analysis is requested from the MSI, computed by the analyser and displayed by the MSI. We have also developed a textual interface that will not be presented in this chapter to allow the usage of the analyser from non-graphical clients. Moreover we will describe an extension of the algorithm used by the tool, originally introduced in [9], which is able to opportunely truncate models with infinite state space. We finally show how the extended algorithm efficiently computes the solution of non-homogeneous models.

In Chapter 6 we consider another way of reducing the complexity of the solution of cooperating stochastic models, i.e., state-space aggregation. In particular, we define a compositional approach to the problem, which has less strict requirements with respect to the state of the art. In Chapter 7, we will show how aggregation and product-form decomposition techniques are indeed related, as we will introduce

a novel concept of *conditional product form*, based on the reversibility of the aggregation of reversed processes. This is the core chapter of the thesis, and the one that justifies the juxtaposition of the previous two chapters. Finally, in Chapter 8 we define and evaluate a method to approximate the solutions of models for which exact aggregation is not possible or feasible.

## Case Studies and Applications

In Part III we show some applications of the solution of cooperating stochastic models, namely on multi-formalism modelling and on a class of product-form Petri nets.

In Chapter 9, we show how product-form solution theory easily couples with multiformalism compositional modeling techniques, to obtain a modeling and analysis framework that offers modeling flexibility and efficient solutions. The contribution is based on the design and implementation of an extensible modeling and solution framework, supported by a tool solving multiformalism Markovian models with a threefold solution mechanism. The tool automatically verifies and performs a product-form solution. If this is not available it provides a state space based analytical solution or a simulation as final backup tool. The research extends the SIMTHESys framework and the tool for product-form solutions, presented in Chapter 5, in order to encompass product form models that satisfy the ERCAT [91] and MARCAT [92] theorems. To date, it appears there is no other similar, tool-supported approach in the current literature.

The aim of Chapter 10, is to analyse a class of SPNs which is useful to model systems in which concurrent activities can lead to conflicts, requiring a recovery phase before a new execution of the same activities is retried. Instances of this kind of systems are quite frequent in the real world, for example in computer networks, databases and operating systems. We provide a formal model for these systems in terms of SPNs and show that they belong to the class studied in [17, 131]. Moreover, we prove two interesting properties for such a class of SPNs: first, their product-form does not require any condition on the transition rates and, second, the joint state space is the Cartesian product of the states that are reachable by each of the model's places. The former property enhances the applicability of the proposed model, while the latter allows us to derive the normalised stationary distribution in a straightforward way for open models. The model that we proposed can be combined with other quasi-reversible components maintaining the product-form property of the joint steady-state distribution.



I

---

**An introduction to stochastic  
modelling**





---

# 1

## Stochastic models

### 1.1 Introduction

In this chapter we give some basic notions about *stochastic models*. As technical details can be easily obtained from textbooks such as [165, 167, 153, 113], we give only an informal outlook on the subject.

A stochastic model is a mathematical abstraction of a system that is characterised by a *stochastic process*. A stochastic process is a set of random variables  $\{X(t) \mid t \in T\}$  defined over the same probability space and indexed by the parameter  $t$ , which usually denotes time. The process random variables take values in set  $\Gamma$ , which is called the *state space* of the process. Both set  $T$  and state space  $\Gamma$  can be either discrete or continuous. If the time parameter  $t$  is continuous or discrete, we have a continuous-time or a discrete-time process, respectively. A discrete-time process can be denoted by  $\{X_n \mid n \in T\}$ , in which the time parameter  $n \in T$  can be seen as a discrete *step*. If the state space  $\Gamma$  is discrete then the process is called discrete-space or *chain*, while it is called a continuous-space process otherwise. The probabilistic behaviour of a 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  $Pr\{X(t_1) \leq x_1; X(t_2) \leq x_2; \dots; X(t_n) \leq x_n\}$ , where  $x_i \in \Gamma$ .

### 1.2 Markovian models

In this thesis we will focus on Markovian (stochastic) processes. A discrete-time Markov process is a process in which the probability of being in a specific state at step  $n + 1$  only depends on the probability of being on a certain state at step  $n$  and is independent of the previous history. The conditional probability distribution of this kind of process satisfies the following condition:

$$Pr\{X_{n+1} = j \mid X_0 = i_0; X_1 = i_1; \dots; X_n = i_n\} = Pr\{X_{n+1} = j \mid X_n = i_n\}, \quad (1.1)$$

for all  $n > 0$ , and  $j, i_0, i_1, \dots, i_n \in \Gamma$ , which is called the *Markov property* for discrete-time processes.

Analogously, a continuous-time process is a Markov process if it satisfies the condition:

$$\Pr\{X(t) = j \mid X(t_0) = i_0; X(t_1) = i_1; \dots; X(t_n) = i_n\} = \Pr\{X(t) = j \mid X(t_n) = i_n\}, \quad (1.2)$$

for all set of times  $t_0 < t_1 < \dots < t_n < t$ , and  $n > 0$ ,  $j, i_0, i_1, \dots, i_n \in \Gamma$ , which is called the *Markov property* for continuous-time processes.

It can be proved that, because of the Markov property, the residence time in each state of the process is distributed according to a distribution with the memoryless property, i.e., the geometric or the negative exponential distribution for discrete-time or continuous-time Markov processes respectively.

First, let us focus on discrete Time Markov chains (DTMCs). If the right-hand side of formula (1.1) is independent of time  $n$ , then a Markov chain is said to be *homogeneous*. In that case, we define the transition probability from state  $i$  to state  $j$  as  $p_{ij} = \Pr\{X_{n+1} = j \mid X_n = i\}$ , and the matrix of transition probabilities  $\mathbf{P} = [p_{ij}]$ , where  $p_{ij} \in [0, 1]$ ,  $\sum_j p_{ij} = 1$ ,  $\forall i, j \in \Gamma$ .

The stationary (or *steady-state*) behaviour of a homogeneous discrete-time Markov Chain can be analysed if the process satisfies some conditions that make it *ergodic*. Informally, a Markov process is said to be *irreducible* if every state can be reached from any other state. Each state can be either *transient*, if there is a non-null probability of never reaching again that state after a given time, or *recurrent*, and it is said to be positive recurrent if the average return time to the state is finite. A state is *periodic* if, after the process is in that state, it can return to the same state only in a number of steps that is multiple of an integer constant  $c \neq 1$ . An ergodic Markov chain is irreducible and formed only by positively recurrent aperiodic states.

Let  $\boldsymbol{\pi} = [\pi_0 \pi_1 \pi_2 \dots]$  be the stationary state probability vector, where  $\pi_j = \Pr\{X = j\}$  is the stationary probability of state  $j \in \Gamma$ , i.e., informally, the probability, on the long run, of observing, at any given moment, that the process is in state  $j$ . Then for homogeneous ergodic discrete-time Markov chains we can compute  $\boldsymbol{\pi}$  as follows [115]:

$$\boldsymbol{\pi} = \boldsymbol{\pi} \mathbf{P}, \quad (1.3)$$

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

We can extend our considerations on DTMCs to continuous-time Markov chains (CTMCs). A CTMCs is homogeneous if the conditional probability on the right-hand side of formula (1.2) is independent of time  $t_n$ , and it only depends on the interval width  $(t - t_n)$ . Thus, we can write the transition probability from state  $i$  to state  $j$ , depending only on the interval width  $s$ , as:

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

Therefore, the state transition probability matrix  $\mathbf{P}(s) = [p_{ij}(s)]$  is width-dependent. We can then define a transition rate matrix  $\mathbf{Q} = [q_{ij}]$ ,  $i, j \in \Gamma$ , which is usually known

as infinitesimal generator, as follows:

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

The steady state behaviour of the continuous-time Markov chain can be evaluated for homogeneous ergodic chain. Notice that checking the ergodicity in a CTMC does not require to verify aperiodicity, since CTMCs are never periodic. The steady state probabilities  $\boldsymbol{\pi} = [\pi_0, \pi_1, \pi_2, \dots]$ , where  $\pi_j = Pr\{X = j\}$  for each state  $j \in \Gamma$ , can be computed by solving the following system of global balance equations:

$$\boldsymbol{\pi}\mathbf{Q} = \mathbf{0}, \tag{1.4}$$

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



---

# 2

## Formalisms and cooperation semantics

In this chapter we introduce some of the basic notions about some of the best known higher level formalisms for stochastic modelling, as well as the semantics of the interaction among components expressed using the same formalism.

### 2.1 Queueing Networks

Queueing networks [20] have been widely used to represent and analyse systems in which resource sharing and, therefore, contention, has to be considered, such as production, communication and computer systems, and have proved to be a powerful and versatile tool for system performance evaluation and prediction. A queueing network (QN) consist of a set of *service centres*, representing system resources, which serve a set of *customers*, which may represent users of a system, requests to a servers, packets to be sent, etc.. Customers compete for the use of the services, and they possibly wait to be served in *queues*, according to a *queueing discipline*. The analysis of isolated queueing centres was originally introduced by Erlang to model congestion of telephonic systems [70], and became popular to model computer networks thanks to Kleinrock's seminal work [116]. Queueing networks extend this class of models allowing customers to move between different service centres according to some *routing* rules [115, 80, 124, 170, 127, 110].

Usually the modeler is interested in analysing a queueing network in order to evaluate a set of performance measures (or indices), such as resource utilisation, throughput and customer response time. The dynamic behaviour of a queueing network can be described by a set of random variables that define a stochastic process. Under some constraints on the network itself, it is possible to define an associated underlying stochastic Markov process, usually a (possibly infinite) CTMC, and to compute the desired performance indices by its solution.

In spite of the constraints needed in order to have an underlying CTMC, and the assumptions that have to be made to keep the model reasonably simple, queueing networks have been proved to be a robust class of models [166].

The analysis and solution of queueing networks, i.e., the computation of performance indices, can be done either analytically or by simulation. While analytical methods offer a theoretically sound analysis of network behaviour, they usually require a set of strict assumptions on both the queueing centres characteristics and on the routing of customers. On the other hand, simulation techniques have a wider range of applicability, but, in order to achieve the desired accuracy, they can be quite onerous to develop and computationally costly to run. Moreover, checking correctness and interpreting simulation outcomes is a difficult task [125].

In order to overcome the aforementioned difficulties in the solution of queueing network models, various paths have been explored in the literature.

Various kind of *product-form* (see Chapter 3 networks where identified, such as Jackson [107], Gordon-Newell [84] and BCMP [26] networks. It is worth noting that, even when the solution exhibits a product-form, the computation of *normalised* steady state probabilities, in the general case, is not a trivial task. The most relevant solution algorithms for closed networks are the Convolution Algorithm [48] and the Mean Value Analysis [151], as well as their extensions for networks in which the customers belong to classes [149, 157, 122, 42]. However, other algorithms for multiclass QNs, with different computational complexities, have been proposed [123, 172, 105, 59, 60, 58, 64].

In the more recent past, research has been focused to the extension of the class of product-form network models and to its characterisation, such as the G-Networks with positive and negative customers [78] that can be used to represent special dynamics of real systems. Other classes of models include various functions of state-dependent routing and several special cases of QNs with finite capacity queues, finite population constraints and blocking [1, 6, 38, 85, 121, 168, 173]. The mathematical properties leading to queueing networks' product-forms are surveyed in [143], while some extensions of product-form QN are presented in [173]. Product-form solution has been extended to networks with batch arrivals and batch services [96, 97] which concern also discrete-time models.

Queueing networks that do not exhibit a product-form solution where also analysed, e.g., Layered Queueing Networks [175, 76] and other types of networks extended to represent more complex system, e.g., with simultaneous resource possession, finite capacity queues and blocking, and fork and join [124, 158, 3, 152, 145, 6].

## 2.2 Stochastic Petri nets

Petri nets (PNs) and their timed extension Stochastic Petri nets (SPNs) [142, 141] are widely used to model concurrent systems in which fork and join synchronisations can occur. Informally, Petri nets are bipartite graphs consisting of places and transitions. Arcs connect places with transitions or vice versa and they are associated with a natural number that represent the weight. When all the arcs have weight 1 we say that the Petri nets is ordinary. A marking associates a natural number

with each place and represents the state of the net. The transitions determine the dynamic behaviour of the net according to the firing semantics that is formally defined in Section 2.2.1. The problem of reachability, i.e., deciding whether, given the initial marking, another marking is reachable, is known to belong to the class of EXPSPACE.

While some timed extensions to Petri nets, such as Stochastic Preemptive Time Petri Nets [43], have a Discrete Time Markov Chain (DTMC) as the underlying stochastic process, here we focus on Stochastic Petri Nets (SPNs), in which each transition fires after an exponentially distributed time that is independent of the firing times of all the other transitions. This implies that the underlying stochastic process is a Continuous Time Markov Chain (CTMC) and the chain's state space corresponds to the state space of the corresponding PN. Once the CTMC is derived, the performance indices can be computed using standard algorithms.

In this section we give some basic notions about *stochastic Petri nets* and *building blocks*, which will be used through the thesis.

### 2.2.1 Stochastic Petri nets

A stochastic Petri net [142, 141] is a tuple,  $\text{SPN} = (\mathcal{P}, \mathcal{T}, \chi(\cdot), \mathbf{I}(\cdot), \mathbf{O}(\cdot), \mathbf{m}_0)$  where:

- $\mathcal{P} = \{P_1, \dots, P_N\}$  is a set of  $N$  places,
- $\mathcal{T} = \{T_1, \dots, T_M\}$  is a set of  $M$  transitions,
- $\chi : \mathcal{T} \rightarrow \mathbb{R}^+$  is a positive valued function that associates a firing rate with every transition; we usually write  $\chi_i$  as an abbreviation for  $\chi(T_i)$ ,
- $\mathbf{I} : \mathcal{T} \rightarrow \mathbb{N}^N$  associates an input vector with every transition,
- $\mathbf{O} : \mathcal{T} \rightarrow \mathbb{N}^N$  associates an output vector with every transition.

A *marking* of the model is a vector  $\mathbf{m} \in \mathbb{N}^N$  that represents the numbers of tokens  $m_i$  in each place  $P_i$ ,  $i = 1, \dots, N$ , and thus it identifies the current state of the model. The initial marking is called  $\mathbf{m}_0$ . A transition  $T_i$  is *enabled* by  $\mathbf{m}$  if  $\mathbf{m} - \mathbf{I}(T_i)$  has non-negative components. An enabled transition  $T_i$  *fires* after an exponentially distributed random time with rate  $\chi_i$ . In this case, the new state  $\mathbf{m}'$  is  $\mathbf{m} - \mathbf{I}(T_i) + \mathbf{O}(T_i)$ . If the input and output vector domains are  $\{0, 1\}^N$ , i.e. tokens move one by one, the net is called *ordinary*.

The graphical representation of SPNs uses circles for places and bars for transitions. If the  $j$ -th component of  $\mathbf{I}(T_i)$  (respectively  $\mathbf{O}(T_i)$ ) is  $k > 0$ , we draw an arc from  $P_j$  (respectively  $T_i$ ) to  $T_i$  (respectively  $P_j$ ) and we label it with  $k$  (for ordinary nets we omit the labels).

The reachability set  $RS(\mathbf{m}_0)$  is the set of all the possible states of the net, given the initial marking  $\mathbf{m}_0$ . In general, the problem of determining the reachability set of a *SPN* is NP-hard and has an exponential space requirement [71]. The

nodes in the *reachability graph* are the states of the reachability set and there is an arc from every node  $\mathbf{m}'$  to  $\mathbf{m}''$  for which there exists a transition  $T$  such that  $\mathbf{m}'' = \mathbf{m}' - \mathbf{I}(T) + \mathbf{O}(T)$ . The incidence matrix  $\mathbf{A}$  of an SPN is an  $M \times N$  matrix, row  $i$  of which is defined as  $\mathbf{O}(T_i) - \mathbf{I}(T_i)$ .

The reachability graph can be either finite or infinite and from it, the continuous time Markov chain (CTMC) underlying the SPN model can be derived simply (either lazily or in a parameterised way if the state space is infinite) [141]. Henceforth we consider models whose underlying CTMCs are ergodic and so admit a unique, equilibrium, state probability distribution. Calculating this can be a difficult computational task because of the state space explosion problem, which causes even a structurally small net to have a reachability set with high cardinality. In such cases, solution of the global balance equations rapidly becomes numerically intractable.

We call the fundamental structure that we use to analyse SPNs in product-form a *building block* (BB). We now formally define a BB and give an expression for its product-form solution, together with sufficient conditions for it to exist.

### 2.2.2 Building blocks

According to [17], a BB consists of a set of places  $P_1, \dots, P_N$ , a set  $\mathcal{T}_I$  of input transitions whose input vectors are null (i.e.  $\mathbf{0} = (0, \dots, 0)$ ), and a set  $\mathcal{T}_O$  of output transitions whose output vectors are null. All the arcs have weight 1. In the BBs for each input transition  $T_y$  there must exist an output transition  $T'_y$  whose input vector is equal to the output vector of  $T_y$ .

**Definition 1** (Building block (BB)). *Given an ordinary (connected) SPN  $S$  with set of transitions  $\mathcal{T}$  and set of  $N$  places  $\mathcal{P}$ , then  $S$  is a building block if it satisfies the following conditions:*

1. *For all  $T \in \mathcal{T}$ , either  $\mathbf{O}(T) = \mathbf{0}$  or  $\mathbf{I}(T) = \mathbf{0}$ . In the former case we say that  $T \in \mathcal{T}_O$  is an output transition while in the latter we say that  $T \in \mathcal{T}_I$  is an input transition. Note that  $\mathcal{T} = \mathcal{T}_I \cup \mathcal{T}_O$  and  $\mathcal{T}_I \cap \mathcal{T}_O = \emptyset$ , where  $\mathcal{T}_I$  is the set of input transitions and  $\mathcal{T}_O$  is the set of output transitions.*
2. *For each  $T \in \mathcal{T}_I$ , there exists  $T' \in \mathcal{T}_O$  such that  $\mathbf{O}(T) = \mathbf{I}(T')$  and vice versa.*
3. *Two places  $P_i, P_j \in \mathcal{P}$ ,  $1 \leq i, j \leq N$ , are connected, written  $P_i \sim P_j$ , if there exists a transition  $T \in \mathcal{T}$  such that the components  $i$  and  $j$  of  $\mathbf{I}(T)$  or of  $\mathbf{O}(T)$  are non-zero. For all places  $P_i, P_j \in \mathcal{P}$  in a BB,  $P_i \sim^* P_j$ , where  $\sim^*$  is the transitive closure of  $\sim$ .*

Note that in an isolated BB, if two or more input (output) transitions have the same output (input) vector, we can fuse them in one transition whose rate is the sum of the rates of the original transitions. Therefore, without loss of generality, we assume that all the input (output) transitions have different output (input) vectors. Finally, to simplify the notation, we use  $T_y$  ( $T'_y$ ) to denote an input (output)



transition, where  $y$  is the set of place-indices of the non-zero components in the output (input) vector of  $T_y$  ( $T'_y$ ). We now recall Theorem 1 that gives sufficient condition for the product-form of a BB.

**Theorem 1** (Theorem 2 of [17]). *Consider a BB  $S$  with  $N$  places. Let  $\rho_y = \lambda_y/\mu_y$ , where  $\lambda_y, \mu_y$  are the firing rates for  $T_y, T'_y \in \mathcal{T}, |y| \geq 1$ , respectively. If the following system of equations has a unique solution  $\rho_i, (1 \leq i \leq N)$ :*

$$\begin{cases} \rho_y = \prod_{i \in y} \rho_i & \forall y : T_y, T'_y \in \mathcal{T} \wedge |y| > 1 \\ \rho_i = \frac{\lambda_i}{\mu_i} & \forall i : T_i, T'_i \in \mathcal{T}, 1 \leq i \leq N \end{cases} \quad (2.1)$$

*then the net's balance equations – and hence stationary probabilities when they exist – have product-form solution:*

$$\pi(m_1, \dots, m_N) \propto \prod_{i=1}^N \rho_i^{m_i}. \quad (2.2)$$

Another interesting result is proved in [131] where the throughput (namely, the reversed rate) of the output transition of a product-form BBs are derived:

**Lemma 1.** *In a product-form BB that satisfies the conditions of Theorem 1, the throughput (reversed rate) of every output transition labelled  $T'_y$  is  $\lambda_y$ , i.e., the rate of the corresponding input transition.*

## 2.3 Markovian Process Algebra

Process algebras are widely used as a formalism for functional analysis of concurrent systems. The main strength of this family of formalisms is the combination of a well-defined semantics and compositionality.

### 2.3.1 Classical process algebras

Process algebras are formal languages, with their own syntax and semantics, which have been widely used for the design and specification of concurrent systems. The most popular and influential process algebras are:

- Milner's Calculus of Communicating Systems (CCS) [137] and its evolution,  $\pi$ -calculus [138].
- Hoare's Communication Sequential Processes (CSP) [104].

Process algebra models (either CCS and CSP) have been used to establish the correctness of concurrent systems. In fact, several qualitative properties can be derived such as liveness.

A process algebra model consists of a set of *agents* that perform atomic actions. The actions can represent sequential behaviours of the agents, communications or synchronisations among them. The major difference between CCS and CSP is on the definition of the semantics for the communication actions. We informally introduce the basic syntax of those calculi and then describe the communication techniques defined for CCS and CSP.

An agent  $P$  is defined according to the following syntax:

$$\begin{aligned}
 P ::= & \quad 0 \\
 & \quad | a.S \\
 & \quad | S + Q \\
 & \quad | S||Q \\
 & \quad | S \setminus M \\
 & \quad | S[a_1/a_0, \dots] \\
 & \quad | 0
 \end{aligned}$$

where  $S$  and  $Q$  are agents,  $a$  and  $a_i$  denote a label and  $M$  is a set of labels.

Informally, the semantics of the operators is the following:

- The prefix operator  $a.S$  models an agent that after performing action  $a$  behaves like  $S$ .
- The choice operator  $S + Q$  says that the agent behaves either as  $S$  or  $Q$ .
- The parallel composition  $S||Q$  denotes that  $S$  and  $Q$  proceed in parallel, possibly communicating with each other.
- The restriction  $S \setminus M$  hides the set of labels  $M$  of  $S$  from outside agents.
- The relabelling  $S[a_1/a_0, \dots]$  replaces in  $S$  the label  $a_0$  by  $a_1$ , more than one replacement can be specified.

Notice that the null agent  $0$  is the agent that cannot do anything and can thus be considered as stuck in a deadlock.

In CCS the communication is defined between pairs of agents. If an agent performs an action  $a$ , then the communication occurs when the other agent performs a complementary action  $\bar{a}$ . The resulting communication action has the special label  $\tau$  that denotes an *internal* action invisible to the environment.

In CSP there is not any concept of complementary action. The synchronization occurs when two agents perform an action with the same label. Here the joint action remains visible to the environment, therefore other concurrent processes can reuse it to communicate with the process just formed by the interaction of the previous ones. This leads to a multiway synchronization.

Stochastic Process Algebras (SPAs) usually have a communication semantics similar to the one of CSP.

CCS and CSP have a *structured operational semantics*. This is based on a labelled transition system (LTS). This allows for the construction of a *derivation graph*, in which the vertices are the language terms and the arcs are the transitions.

A bisimulation is a binary equivalence relation that can be applied to process algebra models. Informally, two systems are bisimilar if they match each other's visible moves. In the bisimulation style of equivalence an agent is characterized by its actions and, in general, the analysis of the derivation graphs is required. If the internal actions are considered observable then we have a *strong equivalence*, otherwise we have a weak equivalence.

As we already stated, both CCS and CSP, being process algebras, allow one to derive several qualitative properties of the modelled systems. In order to perform quantitative analysis, however, timed extensions of these formalisms have been introduced.

### 2.3.2 Process algebras extensions

Since in classical process algebras time is not represented explicitly, performance measures of the models are not possible. The extensions that we name in this section aim to introduce the timing and/or the probabilities of different behaviours in the model description, in order to derive quantitative measures such as expected response time or throughput. In the last years several extensions to process algebras have been defined in order to deal with quantitative analysis. We can identify three classes of extensions:

- Timed process algebras. The idea behind those extensions is to assign a duration to every action, i.e., the operator  $\alpha.P$  becomes  $(\alpha, t).P$  where  $t$  denotes the time required for action  $\alpha$ . These extensions have been proposed for different languages: ACP [4], CSP, CCS and LOTOS [98].
- Probabilistic process algebras. In this case the main idea consists in defining a new semantics for the *choice* operator. Informally,  $S = P + Q$  in a standard process algebra means that process  $S$  can behave either as  $P$  or  $Q$ . In probabilistic process algebras this non-determinism becomes a probabilistic choice, whose details depends on the algebra that is used. Again, the probabilistic extension has been introduced for ACP, CCS, CSP and LOTOS.
- Stochastic process algebras. In this case an action requires a random time to be performed. *Timed processes and performance analysis* (TIPP) is a stochastic process algebra that extends CSP [99]. Action durations are modelled by exponential random variables, therefore the underlying stochastic process is a Markov process. Another process algebra based on a CSP extension is *Performance evaluation process algebra* (PEPA) defined by Hillston in [101].

Even in this case the action duration are exponentially distributed. *Extended Markovian Process Algebra* (EMPA) [31] has been introduced by Bernardo et al. in [30] and provides constructs to represent immediate transitions and non-determinism as well. Stochastic extensions of the CCS family of algebras include the stochastic  $\pi$ -calculus [148] and SPADES [93].

Adding temporal and probabilistic informations to a Process algebra influences the following analysis [98]

- functional behaviour (e.g. liveness or deadlocks),
- temporal behaviour (e.g. throughput, waiting times, reliability),
- combined properties (e.g. probability of timeout).

In this section we focus on Performance Evaluation Process Algebra (PEPA), the most widely used Markovian Process Algebra (MPA). In PEPA every action has a duration that is modelled by an exponentially distributed random variable. The aim of the formalism definition is, using the SOS (Structural Operational Semantics) rules of the algebra, to be able to obtain a Continuous Time Markov Chain (CTMC) given any PEPA model. As for other process algebras, a PEPA model consists of a set of components that can interact according to a small number of combinators: prefix, choice, parallel composition and abstraction. We briefly recall this formal model description in Section 2.3.3. Thanks to the great flexibility of this formalism and the available software tools (see Chapter 4), PEPA has become quite popular in the performance analysis field. However, even a system consisting of simple interacting components may generate a very large CTMC and the exact analysis of the system performance can soon become unfeasible. For this reason, many research efforts have been devoted to study product-forms for PEPA that could possibly allow for more efficient solution algorithms [102, 87, 159, 89, 91]. As we will see in Chapter 3, roughly speaking a product-form model has a steady-state distribution that can be calculated by product of the steady state distributions of its components, even if the components are not stochastically independent. Note that defining efficient solution algorithms for product-models can be a non-trivial task. In Chapter 5 we will show such an algorithm, along with a tool that implements it, while in Chapters 6, 7 and 8 we will show some methods that can be applied even when the model is not in product-form.

### 2.3.3 Performance Evaluation Process Algebra

In this section we introduce PEPA and we recall its syntax. PEPA models are based on the description of component interactions. Each component has an associated set of actions. Let  $\mathcal{Act}$  be the set of all actions, then  $a \in \mathcal{Act}$  is a pair  $(\alpha, r)$  where  $\alpha \in \mathcal{A}$  is the type of the action (and  $\mathcal{A}$  the set of action types) and  $r \in \mathbb{R}^+$  is the

parameter of a negative exponential distribution. In what follows we briefly recall the PEPA operational semantics presented in details in [101].

- The *prefix* combinator models the sequential behaviour, i.e., the component  $(\alpha, r).P$  carries out the activity  $(\alpha, r)$  in  $\Delta t$  time and then behaves as component  $P$ .  $\Delta t$  is an exponentially distributed random variable with parameter  $r$ .
- The *choice* combinator  $+$ , i.e.  $P + Q$ , models a component that behaves as component  $P$  or as component  $Q$ . When a component can perform an activity  $(\alpha, r)$  we say that the activity with type  $\alpha$  is enabled. Let us denote all the activities enabled in  $P$  by  $\mathcal{Act}(P)$ . Then  $\mathcal{Act}(P + Q) = \mathcal{Act}(P) \uplus \mathcal{Act}(Q)$ , where  $\uplus$  denotes the multiset union. The first completed activity determines if the component behaves as  $P'$  or  $Q'$ , where  $P'$  is the component that results from  $P$  completing the activity, and similarly  $Q'$ .
- The *cooperation* combinator models the synchronization and the cooperation among components. We use the following notation:  $P \bowtie_{\mathcal{L}} Q$  where  $\mathcal{L}$  is a set of action types. All the activities in  $P$  and  $Q$  whose type is not in  $\mathcal{L}$  are called individual and are not affected by the operator. On the other hand, the shared activities can be carried on only when they are enabled in both the components  $P$  and  $Q$ . This can cause a component to block waiting for the other. When the activity is enabled in both the components it is carried on with the rate of the slowest. This ensures that  $P \bowtie_{\mathcal{L}} Q$  has an exponentially distributed state resident time. One of the activities contributing to the cooperation can be passive, i.e., it has an unspecified rate  $(\alpha, \top)$ . In this case the other activity determines the rate of the cooperation.
- If  $\mathcal{L} = \emptyset$  the components carry on their activities independently. We call this case pure parallel combinator and we denote it by  $P || Q$ .
- The last combinator is the *abstraction*. We use the syntax  $P/\mathcal{L}$  where  $P$  is a component and  $\mathcal{L}$  a set of types. All the activities in  $P$  with type in  $\mathcal{L}$  cannot be carried out in cooperation with other components, that is they are hidden and assume the unknown type  $\tau$ . However, they still require a time to be completed. The hidden activities can be thought as internal activities of the component.

It is worth recalling that some of the most important recent results about product-forms [89, 91, 92] as well as the seminal idea about component-wise state space aggregations, i.e., the notion of strong equivalence [101], were originally expressed using the PEPA syntax and exploiting the PEPA's cooperation semantics. However, since in this thesis we will not focus on the linguistics aspects of that process algebra, we will recall those theorems, where needed, using a simplified notation.

## 2.4 Stochastic Automata

In this section we introduce another formalism capable of modelling synchronisations between components, i.e., cooperating automata –in a similar fashion of what is done in [46]– however, we restrict our analysis to pairwise cooperations (as, e.g., in [72]). We use bold letters to denote matrices and vectors (which must be considered row-vectors unless differently stated).  $\mathbf{e}_n$  is the  $n$ -dimension vector whose components are all 1,  $\mathbf{I}_n$  is the identity matrix of size  $n \times n$ . Sizes are omitted when they can be implicitly assumed. In what follows, we first introduce the semantics of the synchronisation between two components and then give the restrictions assumed in this thesis.

Let us consider a pair of automata  $M_1$  and  $M_2$  which synchronise on a set of transition types  $\mathcal{T} = \{1, 2, \dots, T\}$ . The rate of a transition type is a positive real number  $\lambda_t$ ,  $t \in \mathcal{T}$ . For each label  $t \in \mathcal{T}$  we define two matrices  $\mathbf{E}_{1t}$  and  $\mathbf{E}_{2t}$  that describe the behaviour of component-automaton  $M_1$  and  $M_2$ , respectively, with respect to synchronising label  $t$  and whose dimensions are  $N_k \times N_k$  for  $\mathbf{E}_{kt}$ , with  $k = 1, 2$  and  $N_k$  representing the number of states of  $M_k$ .

Matrix element  $\mathbf{E}_{kt}(s, s')$  denotes the probability that automaton  $M_k$  moves from state  $s$  to state  $s'$  joint with a transition with the same type  $t$  performed by the other automaton; hence  $1 \leq s, s' \leq N_j$ , and  $0 \leq \mathbf{E}_{kt}(s, s') \leq 1$ . Moreover, the sum  $R_{kt}(s)$  of any row  $s$  of matrix  $\mathbf{E}_{kt}$  is in the interval  $[0, 1]$  and can be interpreted as the probability that component  $M_k$  accepts to synchronise on  $t$  given that its actual state is  $s$ .

The infinitesimal generator  $\mathbf{Q}$  of the CTMC underlying the synchronisation of the two automata is defined as follows [164]:

$$\mathbf{Q} = \sum_{t=1}^T \lambda_t (\mathbf{E}_{1t} \otimes \mathbf{E}_{2t}) - \sum_{t=1}^T \lambda_t (\mathbf{D}_{1t} \otimes \mathbf{D}_{2t}), \quad (2.3)$$

where  $\mathbf{D}_{kt} = \text{diag}(\mathbf{E}_{kt} \mathbf{e}^\top)$ , and  $\text{diag}(\mathbf{v})$  (with  $\mathbf{v}$  a  $n$ -dimension row-vector) is defined as the  $n \times n$  matrix:

$$\text{diag}(\mathbf{v})(s, s') = \begin{cases} \mathbf{v}(s) & \text{if } s = s' \\ 0 & \text{otherwise} \end{cases},$$

and  $\otimes$  denotes the Kronecker's product operator.

Note that if model  $M_1$  is such that  $\mathbf{E}_{1t_1} = \mathbf{E}_{1t_2} = \mathbf{I}$  with  $t_1 \neq t_2$ , then we can replace transition types  $t_1$  and  $t_2$  with a new type  $t^*$  for which  $\mathbf{E}_{1t^*} = \mathbf{I}$  and  $\mathbf{E}_{2t^*} = (\lambda_{t_1} \mathbf{E}_{2t_1} + \lambda_{t_2} \mathbf{E}_{2t_2}) \lambda_{t^*}^{-1}$ , where  $\lambda_{t^*} = \max_s (\sum_{s'} (\lambda_{t_1} \mathbf{E}_{2t_1} + \lambda_{t_2} \mathbf{E}_{2t_2})(s, s'))$ . As a consequence, we order the transition types such that: for  $t = 1$ , we have  $\mathbf{E}_{21} = \mathbf{I}$ , for  $t = 2$  we have  $\mathbf{E}_{12} = \mathbf{I}$  whereas for  $2 < t \leq T$  we have the remaining non-trivial synchronising transition. The reason for distinguishing  $t = 1, 2$  and  $t > 2$  will be clear later on.

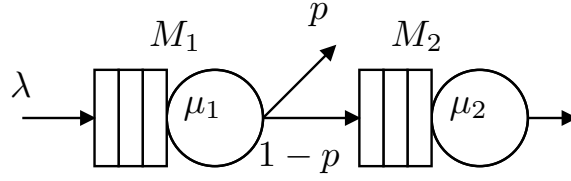


Figure 2.1: Tandem of exponential queues.

In order to familiarise with the proposed notation, we model a simple tandem of two exponential queues by means of automata and discuss the advantages of the choice of this formalism with respect to other possible candidates.

**Example 1** (Automata representation). *Note that this model representation removes the asymmetries that arise in some Markovian process algebra (e.g. PEPA) between active and passive transitions. To see this, consider a simple tandem of two queueing stations with independent, exponentially distributed, service time with mean  $1/\mu_1$  and  $1/\mu_2$ . Customers arrive from the outside according to a homogeneous Poisson process with rate  $\lambda$ . After being served at the first queue, customers may enter the second one with probability  $1-p$  or leave the network with probability  $p$  (see Figure 2.1). In this example, the independent transitions of  $M_1$  ( $t = 1$ ) correspond to the arrival of customers and to the service completions at the first queue after which the customers leave the network, whereas the independent transitions of  $M_2$  ( $t = 2$ ) correspond to the service completions at the second queue. According to the notation introduced, we have:*

$$\mathbf{E}_{11} = \begin{pmatrix} 0 & 1 & 0 & 0 & & \\ \gamma & 0 & 1 & 0 & \dots & \\ 0 & \gamma & 0 & 1 & & \\ \vdots & & \ddots & & \ddots & \end{pmatrix}, \quad \mathbf{E}_{12} = \mathbf{I}, \quad \mathbf{E}_{13} = \begin{pmatrix} 0 & 0 & 0 & 0 & & \\ 1 & 0 & 0 & 0 & & \\ 0 & 1 & 0 & 0 & \dots & \\ & & \ddots & & & \end{pmatrix},$$

$$\mathbf{E}_{21} = \mathbf{I}, \quad \mathbf{E}_{22} = \begin{pmatrix} 0 & 0 & 0 & 0 & & \\ 1 & 0 & 0 & 0 & \dots & \\ 0 & 1 & 0 & 0 & & \\ & & \ddots & & & \end{pmatrix}, \quad \mathbf{E}_{23} = \begin{pmatrix} 0 & 1 & 0 & 0 & & \\ 0 & 0 & 1 & 0 & \dots & \\ 0 & 0 & 0 & 1 & & \\ \vdots & & & & \ddots & \end{pmatrix},$$

with  $\lambda_1 = \lambda$ ,  $\gamma = \mu_1(1-p)/\lambda$ ,  $\lambda_2 = \mu_2$ , and  $\lambda_3 = \mu_1 p$ . As mentioned before, in the SAN master/slave synchronisation [146] or in the PEPA active/passive cooperation [101], the modeller has to choose for each synchronising type which automaton is active (and hence determine the rate of the transition in the joint model) and which is passive (i.e., is forced to move at the rate imposed by the active). Using the formalism proposed in [46] there is no need for this distinction.





---

# 3

## Product Forms

As we have already seen, Markovian models have proved to be a robust and versatile support for the system performance analysis community. Performance modelling and analysis of complex heterogeneous systems and networks based on analytical model is usually done on an abstraction described using a high-level formalism, such as Stochastic Petri Nets (SPNs), Performance Evaluation Process Algebra (PEPA), queueing systems or networks, that have an underlying Continuous Time Markov Chain (CTMC). The desired performance indices, under steady-state conditions, are computed by the evaluation of the stationary state probabilities of the CTMC. This computation is usually a hard task, when not unfeasible, since it requires the solution of the system of Global Balance Equations with a computational time cost of  $O(Z^3)$ , where  $Z$  denotes the number of ergodic states of the model. Under special conditions it is possible to define some efficient algorithms for numerical (exact or approximated) solution of the GBEs.

Product-form models take a different approach. They apply the *divide et impera* paradigm to efficiently solve complex models. Informally, a model  $S$  is seen as consisting of a set of  $N$  interacting sub-models  $S_1, \dots, S_N$ . A state of  $S$  can be defined as  $\mathbf{m} = (m_1, \dots, m_N)$  where  $m_i$  is a state of  $S_i$ .  $S$  is in product-form with respect to  $S_1, \dots, S_N$  if its stationary distribution  $\pi(\mathbf{m})$  satisfies the following property:

$$\pi(\mathbf{m}) \propto \prod_{i=1}^N g_i(m_i),$$

where  $g_i$  is the stationary distribution of sub-model  $i$  considered in isolation and appropriately parametrised. Roughly speaking, from the point of view of a single sub-model, the parametrisation abstracts out the interactions with all the other sub-models. Since the state space of a sub-model  $S_i$  is much smaller than that of  $S$  the solution of its GBEs may be computed efficiently. Modularity becomes a key-point both for the analysis and the description of the model, since it is a good engineering practice to provide modular models of systems.

Exploiting the product-form solutions requires to address two problems:

1. Deciding whether model  $S$  is in product-form with respect to the given sub-models  $S_1, \dots, S_N$ ;

2. Computing the parametrisation of the sub-models  $S_1, \dots, S_N$  in order to study them in isolation.

We have not listed the solution of the sub-model CMTCs as a problem because we suppose that the cardinalities of their state spaces are small enough to directly solve the GBEs. If this is not the case, a product-form analysis of the sub-models may be hierarchically applied. In literature, Problem 1 has been addressed in two ways. The first consists in proving that a composition of models that yield some high-level characteristics is in product-form. For instance the BCMP theorem [26] specifies four types of queueing disciplines with some service properties and states that, under certain conditions, a queueing network of such models has a product-form solution. The second approach is more general, i.e., the properties for the product-form are defined at the CTMC level. Although this can lead to product-form conditions that are difficult to interpret, this approach really enhances the compositionality of the models, because it is possible to combine sub-models originated by different high-level formalisms. In this chapter, we often refer to a recent result about product-forms: the Reversed Compound Agent Theorem (RCAT) [89]. This theorem has been extensively used to prove a large set of product-form results previously known in literature (BCMP product-form [88], G-networks with various types of triggers [90], just to mention a few). Problem 2 is usually strictly related to Problem 1. In general, the parametrisation of the sub-models requires the solution of a system of equations that is called system of traffic equations. For several years it was assumed that product-form solutions must be derived from linear systems of traffic equations, but the introduction of G-networks has shown that this is not necessary.

In this section we provide the basic theoretical background and the notation needed to understand the content of Chapter 5. Most of the topics will be introduced informally, but the interested reader may refer to the original works for more precise and detailed information.

The algorithm that is described in Section 5.3 is based on the product-form analysis introduced by Harrison in [89]. The following paragraphs briefly sketch these results, using a simplified notation, after giving an introduction to the class of models we refer to.

**Model description and interaction.** Consider a finite set of models  $S_1, \dots, S_N$ . We denote by  $n_i, n'_i, n''_i \dots$  the states belonging to the state space of  $S_i$ . Transitions may change the state of model  $S_i$  and are characterised by:

- a label  $a_i$
- a departing state  $n_i$  and an arrival state  $n'_i$ , we write  $n_i \xrightarrow{a_i} n'_i$
- an active transition with rate  $q_i(n_i \xrightarrow{a_i} n'_i)$ , i.e., the positive parameter of the exponential random variable associated with the time required by that transition to be carried out. Alternatively, the transition may be *passive*, i.e., its rate is undetermined and we denote this by the  $\top$  symbol.

Note that it is required that all the transitions with the same label in  $S_i$  are either active or passive. Let  $\mathcal{A}_i$  denote the set of active labels for  $S_i$  and  $\mathcal{P}_i$  that of the passive labels. The synchronisation between two models  $S_i$  and  $S_j$  occurs when there is a label  $a \in \mathcal{A}_i \cap \mathcal{P}_j$  or  $a \in \mathcal{P}_i \cap \mathcal{A}_j$ . In the former case  $S_i$  is the active model with respect to the synchronisation  $a$ , in the latter it is the passive (and vice versa for  $S_j$ ). When  $a$  is a synchronising label for models  $S_i$  and  $S_j$ , these perform the transitions labelled by  $a$  only jointly. The rate of the joint transition is defined by the rate of the active model. Note that we are defining pairwise synchronisations. A formal analysis of this semantics can be found in [101, 89]. A final assumption is that if  $a \in \mathcal{A}_i$ , for each state in  $S_i$  there is a finite number of outgoing arcs labelled by  $a$  and if  $a \in \mathcal{P}_i$  for each state in  $S_i$  there is only one outgoing arc labelled by  $a$ . This condition is trivially satisfied for all the well-known product-form stochastic models.

**RCAT formulation.** Since a model  $S_i$  may have passive transitions, its infinitesimal generator in isolation cannot be computed, and hence its equilibrium probabilities are unknown. Roughly speaking, RCAT gives us a way to compute a value  $K_a$  associated with each passive label  $a$ . Once these are known, we can modify model  $S_i$  such that all the transitions labelled by  $a \in \mathcal{P}_i$  take  $K_a$  as a rate (and we write  $S_i^c = S_i\{a \leftarrow K_a\}$ ). This modified model is called  $S_i^c$ , where the  $c$  stands for closure. Then, the stationary distribution of the cooperating model is the product of the stationary distributions of the closure of its sub-models  $S_1^c, \dots, S_N^c$  if  $K_a$  is the reversed rate of all the transitions labelled by  $a$  in  $S_j^c$  such that  $a \in \mathcal{A}_j$ . The formulation of the following theorem is slightly different from the original [89] mainly because we use a different notation (in the original paper a Markovian process algebra is used). In what follows we state RCAT extended to multiple pairwise interactions and adapted to the notation we introduced.

**Theorem 2** (RCAT [89]). *Given a set of cooperating models  $S_1, \dots, S_N$  assume that the following conditions are satisfied:*

1. *for all  $i$ , if  $a \in \mathcal{A}_i$  then for each state  $n_i$  of  $S_i$  there is exactly one state  $n'_i$  such that  $n'_i \xrightarrow{a} n_i$*
2. *for all  $i$ , if  $a \in \mathcal{P}_i$  then for each state  $n_i$  of  $S_i$  there is exactly one state  $n'_i$  such that  $n_i \xrightarrow{a} n'_i$*
3. *there exists a set of positive values  $\mathcal{K} = \{K_a, a \in \mathcal{A}_i \cap \mathcal{P}_j, i, j = 1, \dots, N\}$  such that when all the models are closed using the values in  $\mathcal{K}$  we have that  $K_a \in \mathcal{K}$  is the rate of all the transitions labelled by  $a$  in the reversed process of  $S_i^c$ , where  $a \in \mathcal{A}_i$ .*

*Then, the stationary distribution of positive recurrent state  $\mathbf{n} = (n_1, \dots, n_N)$  is in*

product-form:

$$\pi(\mathbf{n}) \propto \prod_{i=1}^N \pi_i(n_i),$$

where  $\pi_i(n_i)$  is the stationary distribution of state  $n_i$  in  $S_i^c$ .

We refer to conditions 1 and 2 of Theorem 2 as *structural conditions*, and to condition 3 as the *traffic equation condition*. Note that a key-concept in the formulation of Theorem 2 is the rate of the active transitions in the reversed processes of the closed sub-models. If we have a transition  $n_i \xrightarrow{a_i} n'_i$ , with  $a_i \in \mathcal{A}_i$  and  $n_i, n'_i$  states of  $S_i$ , then the following relation holds [111, 89]:

$$\bar{q}_i(n_i \xrightarrow{a_i} n'_i) = \frac{\pi_i(n_i)}{\pi_i(n'_i)} q_i(n_i \xrightarrow{a_i} n'_i), \quad (3.1)$$

where  $\bar{q}_i(n_i \xrightarrow{a_i} n'_i)$  denotes the rate of the transition from  $n'_i$  to  $n_i$  labelled by  $a_i$  in the reversed process corresponding to  $n_i \xrightarrow{a_i} n'_i$  in the forward one.

---

# 4

## Tools

### 4.1 Introduction

As we already noted, performance and reliability evaluation of computer systems and networks is a widely studied topic and a powerful tool for system designers and maintainers. However, for a practitioner in the software or hardware engineering field, the modelling phase required by those kinds of analysis could represent a showstopper, because commonly used formalisms can be difficult to understand and because they are suitable and designed for specific and restricted classes of problems. For example, while Queueing Networks [20] are an easy to learn formalism to model resource contention, Stochastic Petri Networks [141], are better suited for modelling synchronisation in concurrent settings. Hence performance engineers and designers would take a great advantage from the availability of a set of different formalisms in performance modelling tool, to exploit the different potentials for modelling and analysis of complex computer systems.

However, the differences in the semantics of formalisms often poses significant difficulties in defining how they should integrate. Moreover, usually system performance evaluation can be carried on through different modelling solution techniques and algorithms, e.g., symbolic or numerical analysis, which could be exact or approximate, and simulation. The various modelling solution methods should be included in the tools and available to the user, along with the statistical information on the quality of the result obtained. Notice, however, that in general there is a tradeoff between the expressivity of a formalism and the computational complexity required for the exact derivation of the desired performance/reliability indices. Finally, the availability of an intuitive user interface is of paramount importance. Ideally, a user should choose between a Graphical User Interface (GUI), which allows one to rapidly design complex systems, and a textual, command line interface or a programming language, in order to automatise repetitive tasks and to feed the tool with data generated automatically by other software.

The purpose of this chapter is to analyse software packages that allow analysts to express and solve models using more than one formalism. Having introduced some of the difficulties that the development of such a software could encounter, it should not be a surprise that very few of them actually exists. The motivation behind this

analysis is to put the tool described in Chapter 5, and in particular its integration with a multi-formalism designing tool, described in Chapter 9. . At the best of our knowledge, a surveying activity on this topic was never attempted before. However, in [109] the authors gives an overview on tools for evaluating reliability, availability and serviceability available at the time.

This chapter does not focus neither on design methodologies that allow for performance analysis during the software development cycle nor on the mathematical basis underlying modelling formalisms. A survey on the former topic is presented in [19], while for the latter the reader can refer to [32].

**Software classification** In order to classify the analysed software packages, it is necessary to determine some qualitative criteria. In this chapter we consider the following characteristics:

- The type of formalisms used.
- The kind of solutions obtainable by the tool.
- The kind of user interface.
- The amount of available documentation.
- The license of the software.
- The maintenance status of the software.
- The hardware and software platforms supported.

Other important characteristics of the software, such as the level of interoperability with other software tools, the ease of use or the pricing scheme are not considered here, because they need a subjective evaluation, which is out of the scope of this document.

**Single-Formalism Tools** Over the years, many tools able to model and analyse systems expressed in a single formalism have been proposed and implemented. Restricting the modelling expressiveness often allows one to exploit formalisms' features that permit faster or more precise solutions. As we are focusing on multiple-formalism softwares, here we recall only some of the most popular or influential single-formalism performance design tool in the literature. We can roughly distinguish the design formalisms, and thus tools, in the following categories:

- **Markov Processes:** this is the class of formalisms at the lowest level of abstraction usually employed for performance evaluation. Most of the higher-level formalism used in that field is based on them. As the number of states increases, advanced software techniques are required in order to maintain numerical stability and a reasonable space and time computational complexity,

e.g., exploiting the presence of regularities in the model specification. Tools that solve Markov or Semi-Markov processes [113, 154], given some structural assumptions, include SMCSolver [34]. Some extension to classical continuous time Markov chains in which it is possible to synchronise transition between different chains, such as Stochastic Automata [146] or formalisms derived from process algebras, have been proposed. A software that allows for the solution of such models is described in [15].

- **Stochastic or Probabilistic Process Algebras:** those formalisms, such as PEPA [101] describe a system in term of interacting processes specified using a formal language with a well-defined semantics, on which algebraic manipulations and model checking are possible. Tools that allow models written using process algebras to be analysed include the PEPA Eclipse Plugin [169], the Imperial/International PEPA Compiler [41], the PEPA workbench [82], ipc/HYDRA [40] and PRISM [119].
- **Queueing Networks:** they are among the oldest and most used class of formalism for performance modelling and evaluation. Unfortunately, while their use can be fairly intuitive, their lack of a formal and simple semantics makes the implementation of all their possible variants difficult for a software designer point of view. Software packages that implement some classes of queueing networks include the qnetworks toolbox [135], QMAM [144], QNAP2 [174], FiFiQueues [155], PEPSY-QNS [114] and the Java Modelling Tools [33]. Layered Queueing Networks [76], an extension of this class of formalism, can be analysed using the Layered Queueing Network Solver. SPE·ED [162] is a tool for the Software Performance Engineering (SPE) [161] where the behaviour of software is modelled as an *execution graph*, on which the tool can perform a static analysis and translate it in a queueing network.
- **Petri Nets:** Variants of Petri Nets that take into account time and probabilistic behaviour, e.g., Stochastic Petri Nets, Generalised Stochastic Petri Nets [134] or Stochastic Activity Networks [136], have proved to be a popular choice among performance evaluation software designers. Among the tools that can be used to analyse models designed using this class of formalisms we can find TimeNET [177], PIPE [36, 68, 37], DSPNExpress [129], SPNP [103] and GreatSPN [51]. A hybrid formalism named Queueing Petri Nets [27], which combines Coloured GSPNs with queues, is used by QPME [118]. Another extension to Petri Nets, the Abstract Petri Net Notation (APNN) [28], was proposed in order to have a formalism capable to encompass all the possible features of common formalisms used in performance evaluation. APNN Toolbox [47] is a tool which allows the analysis of systems modelled using APNNs. The DEDS (Discrete Event Dynamic Systems) Toolbox [29] allows the analyst to design tools that support multiple formalisms through APNN

as a common intermediate one. To the best of our knowledge the tool is no more publicly available, and thus has not been included in this chapter.

The list above does not cover all the possible input formalisms used by software tools. Less popular formalisms have also been exploited, such as the ad hoc specification languages used by Mosel [57] and Modest/Motor [35], or probabilistic temporal logics used by model checkers such as the already-cited PRISM. Although those formalisms could be useful in specific environment, and they can help practitioners coming from other fields to include performance analysis in their work, their scarce diffusion among the performance evaluation community makes their use and acceptance somewhat more problematic.

## 4.2 Software Analysis

In this section we analyse some of the softwares that allow one to design system models combining different formalisms. Although some early efforts on multi-formalism design tools development are mentioned in the literature, e.g., the Integrated Modelling Support Environment (IMSE) [147] and DNAmaca [117], we focus only on software packages for which we were able to verify the existence of their implementation and that appear to have been in active development at least after the year 2000.

**SHARPE** SHARPE (Symbolic Hierarchical Automated Reliability and Performance Evaluator) [156], developed at the Duke University since 1986 [171], is a software that allows one to specify and analyse performance, reliability and performability models using a variety of formalisms. SHARPE provides a specification language that can be used both interactively or to write scripts (*batch mode*). Moreover, a graphical user interface is available. Each modelling formalism has keywords in the language in order to specify the design of the model.

Supported formalisms include Markov and semi-Markov chains, Markov regenerative processes Single and Multi-Chain Product Form Queueing Network, i.e., closed queueing networks whose nodes belong to the set of types defined for BCMP networks [26], Generalised Stochastic Petri Nets (GSPNs) , Stochastic Reward Nets, Reliability Block Diagrams (RBDs), fault trees, reliability graphs, and series-parallel graphs [170]. For each of these formalisms, a set of corresponding solution functions can be invoked, in order to get numerically-computed values for performance parameters. Different solution methods, both for transient and steady state analysis, are used, depending on the input formalism. Submodels written in different formalisms can be composed hierarchically, even if not all formalisms, most notably queueing networks or GSPNs, can benefit from this feature. Part of a composed model, drawn from the example library of the software, is shown in Figure 4.2, while graphing capabilities can be seen in Figure 4.4. SHARPE's supported platforms include Solaris,



Windows and Linux. SHARPE is a proprietary software, but it is free for academic users. The software is mature and well documented, and its latest release at the time of the writing of this chapter was released, for Windows only, on February 2010. Other tools, such as the Software Reliability Estimation and Prediction Tool (SREPT) [150], can use SHARPE as a component to solve some class of models.

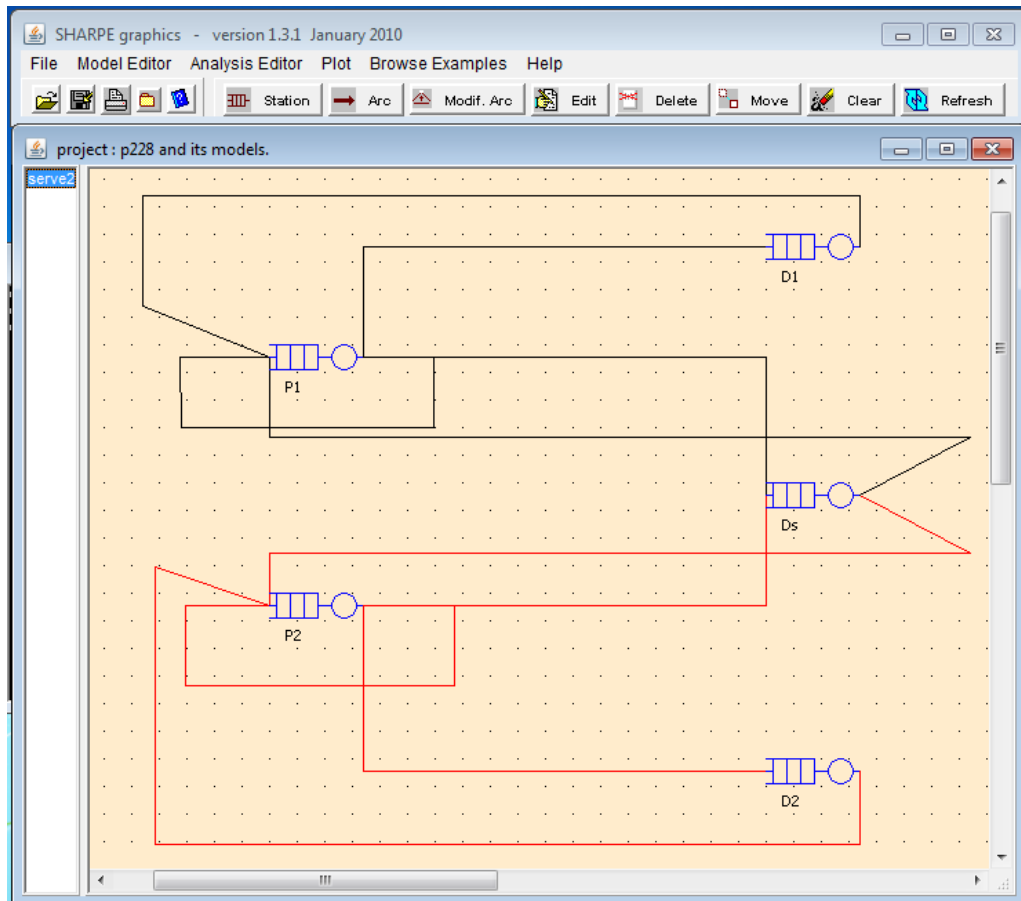


Figure 4.1: SHARPE: a Multiclass Product Form Queueing Network

**Möbius** Möbius [54], developed at the University of Illinois, is a tool for modelling complex systems using various formalisms, such as Stochastic Activity Networks (SAN) and their simplified version, called *Buckets and Balls*, PEPA and its extension  $PEPA_k$  [55], Fault Trees. Each formalism can be used to create *atomic* models, i.e., models described using a single formalism. Models consist of *state variables* and *actions*, representing transitions among states. *Composed* models can be made by combining atomic models that use different formalisms in order to describe complex systems made up of different components. Once a model is built, in order to specify performance measures it is necessary to have a *reward model*, that associates *reward*

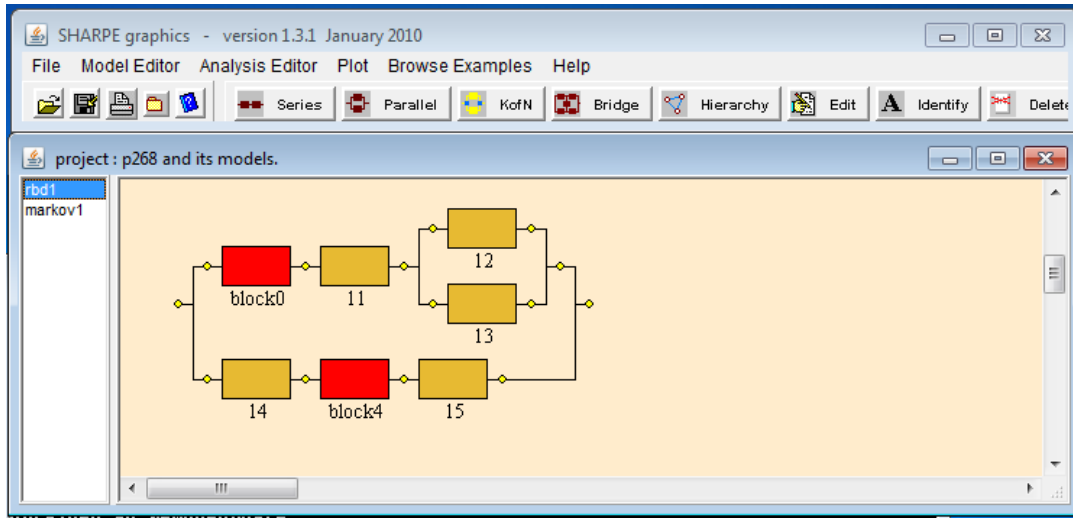


Figure 4.2: SHARPE: composed model, RBD

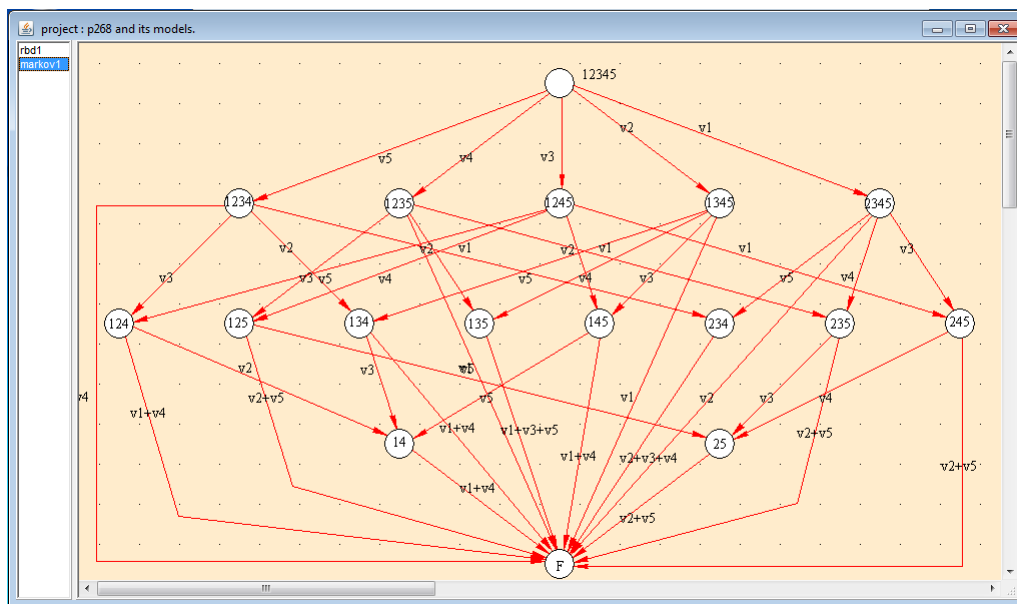


Figure 4.3: SHARPE composed model, Markov chain

*variables* with a composed model. It is then possible to use a *solver* to compute the values of those variables, i.e., a *result*.

There are two kinds of solver: *numerical solvers* and *simulators*. Numerical solvers are able to find the exact values (within a numerical precision) of performance variables. However, the applicability of those solvers is limited to models whose underlying stochastic processes are Markovian or deterministic with a finite state space. In order to compute this state space, a *transformer* is used. Those transformers can generate a flat state space exploring the process or a symbolic one,

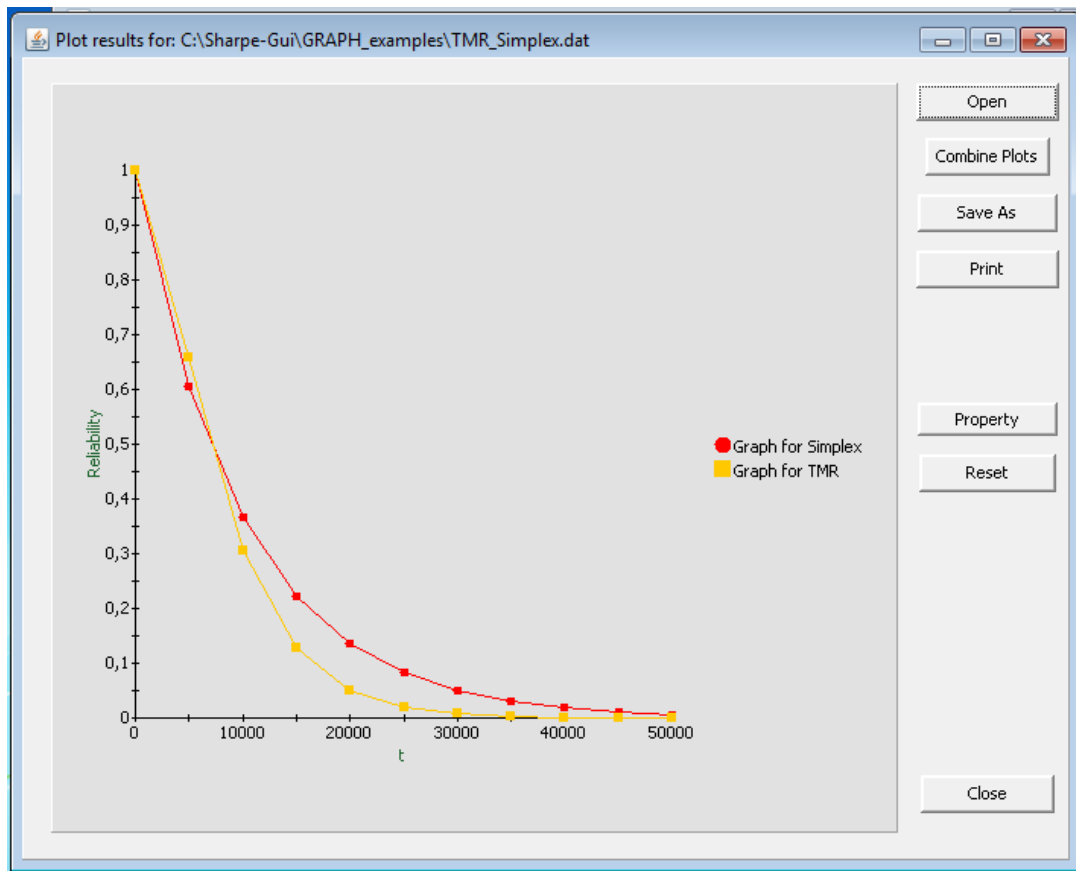


Figure 4.4: Plotting capabilities of SHARPE

i.e., a compact representation of the space state, obtained using lumping techniques. Once the state space is generated, the chosen numerical solver computes the steady state probabilities of the model. The Möbius' simulator can be used with every kind of model that can be expressed using the software itself and, using discrete event simulation, it can find both transient and steady state solutions for the model. Moreover, the simulation can be distributed among different computers, parallelising the computation. However simulation results can suffer from statistical and numerical problems [125]. Solutions are provided with confidence intervals. Solutions found using different techniques can be combined into *connected model solvers*, where, in a multi-step approach, the solutions of separate models are combined using *connection functions* in order to compute the global solution. Moreover, solution data of multiple experiments can be analysed using a *study*.

Möbius is equipped with a graphical user interface, that is used both to design models and to get and analyse solutions, although many of its components can be also executed in a non-interactive fashion using the command line. A screenshot from an example composed model available in Möbius is shown in Figure 4.5, while the output of a simulation run on the same model is shown in Figure 4.8.

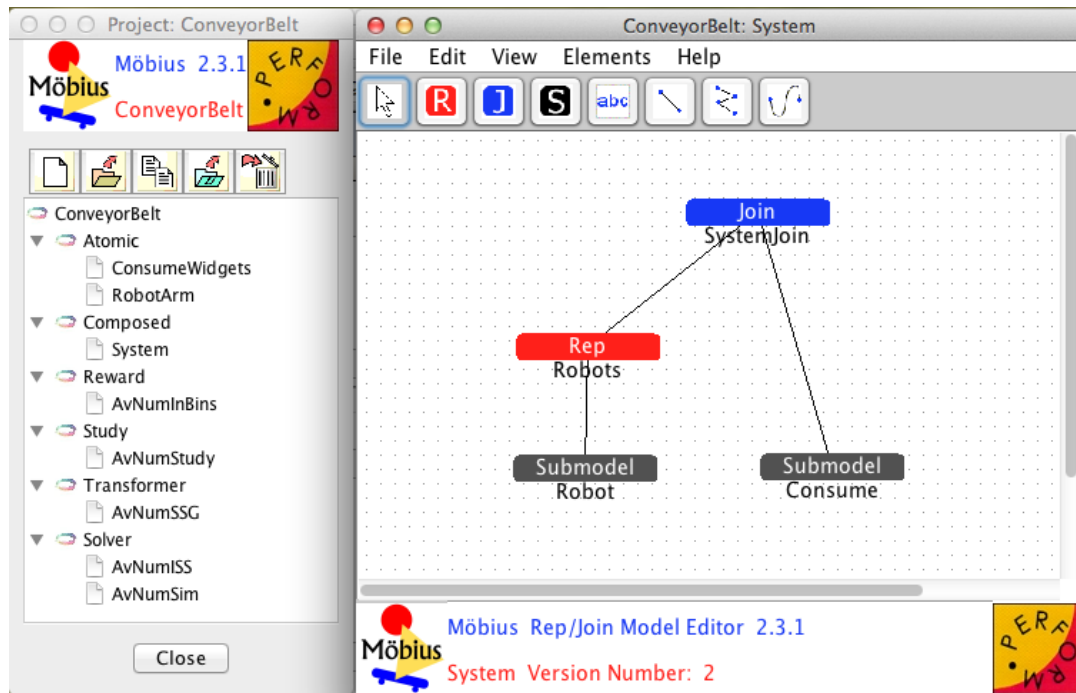


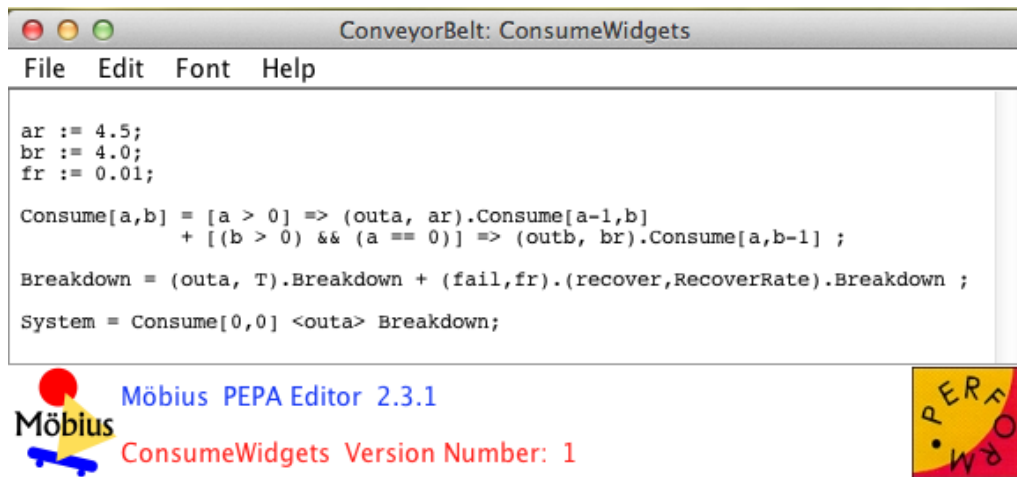
Figure 4.5: Möbius: a composed model

The tool is written in Java and supported platforms include Linux, MacOS X and Windows. Möbius is a proprietary software, but it is free for academic non-profit users. The software is mature, it appears to be regularly updated and maintained, and provides an extensive documentation of the features implemented. The last release, as we write those words, is numbered 2.3.1.

**SMART** SMART (Stochastic Model checking Analyzer for Reliability and Timing) [52] is a software tool that allows one to model complex systems and to analyse them using both performance analysis and symbolic model checking techniques. Currently the formalisms supported are Markov Chains (both at discrete and continuous time) and Stochastic Petri Nets, however the authors plan to add more formalisms in the future.

SMART provides a language to specify the behaviour of the system. The language supports features like cycles and recursion, which allow the user to write algorithms. From the performance evaluation perspective, SMART provides numerical solution algorithms for transient and steady-state analysis of stochastic processes. Discrete event simulation is mentioned in the documentation but it seems that it is not yet completely implemented. The literature on the tool shows a particular focus on enhancing the efficiency of the tool, e.g., storing large state spaces in a compact representation [53, 139].

The software is available for Linux, Mac OS X and Windows. The licensing



```

ConveyorBelt: ConsumeWidgets
File Edit Font Help

ar := 4.5;
br := 4.0;
fr := 0.01;

Consume[a,b] = [a > 0] => (outa, ar).Consume[a-1,b]
+ [(b > 0) && (a == 0)] => (outb, br).Consume[a,b-1] ;

Breakdown = (outa, T).Breakdown + (fail,fr).(recover,RecoverRate).Breakdown ;

System = Consume[0,0] <outa> Breakdown;

```

Möbius PEPA Editor 2.3.1  
ConsumeWidgets Version Number: 1

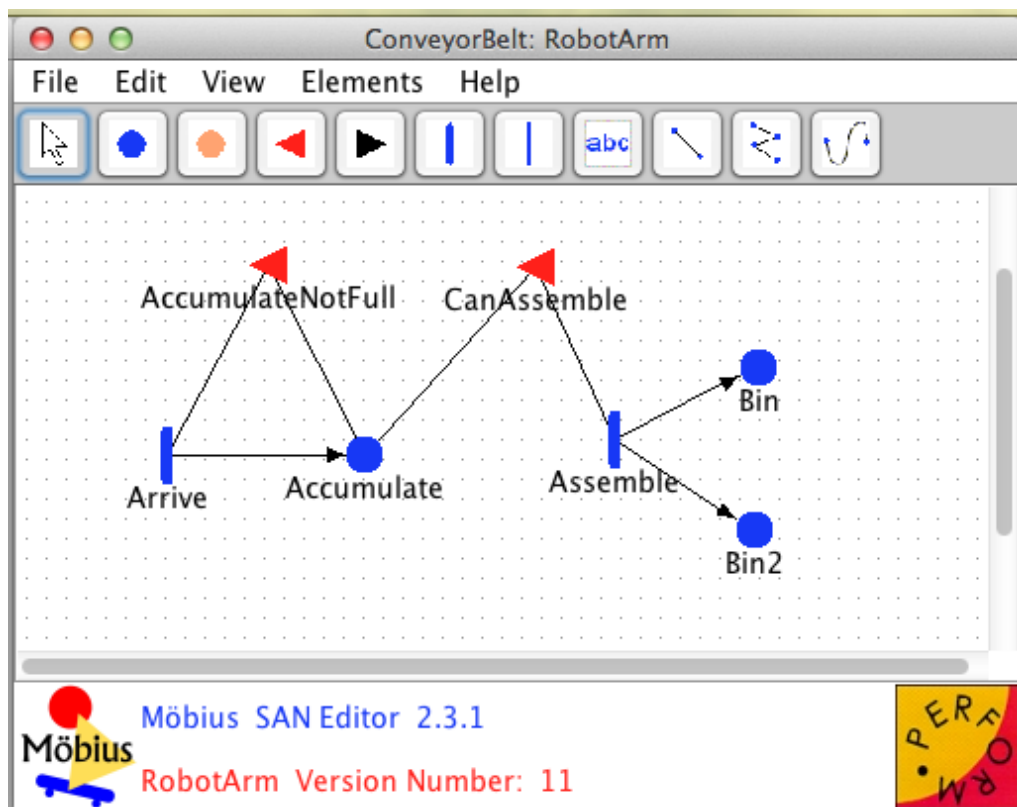
Figure 4.6: Möbius: a PEPA<sub>k</sub> submodel

Figure 4.7: Möbius: a SAN submodel

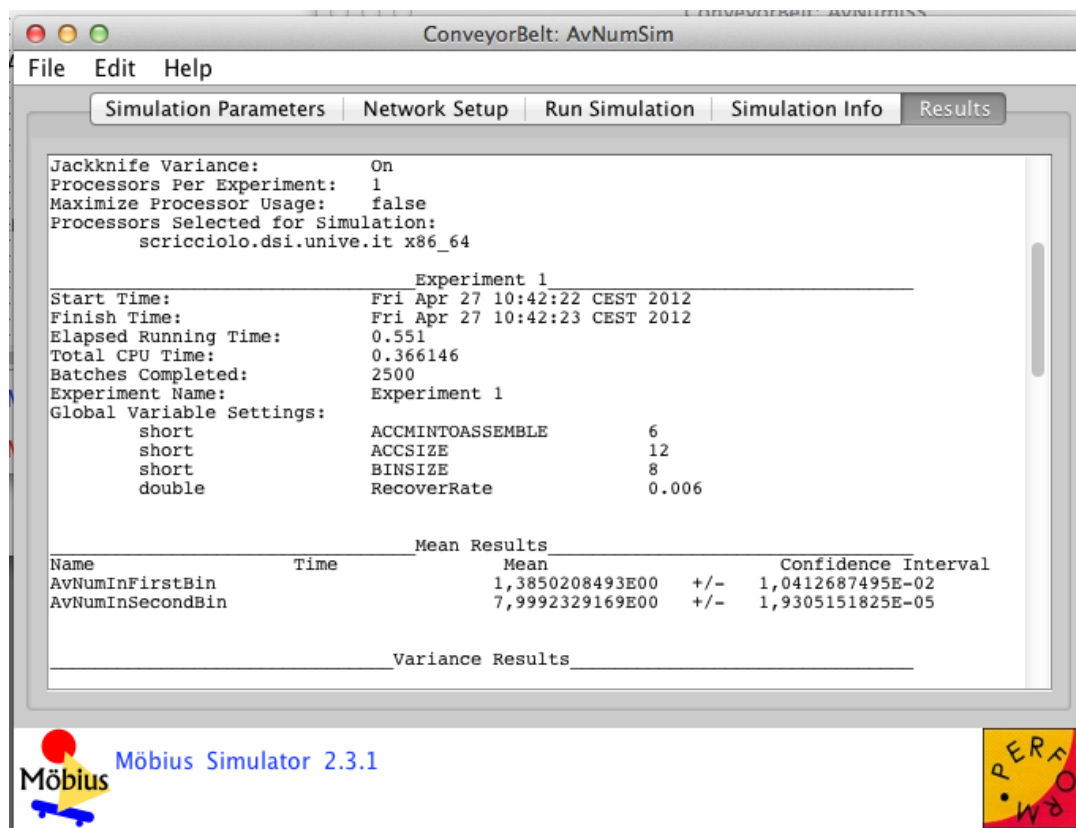


Figure 4.8: Möbius: simulation results

scheme is not stated in the home page of the projects, and a private inquiry has to be made in order to get the software. At the time of the writing of this chapter, we were not able to test the tool. However an extensive documentation is freely available, and we have based our considerations on it. The first prototype of the software was built between 1994 and 1996. The last public activities on the project appears to be from 2008.

**SIMTHESysEr** SIMTHESysEr is a software, based on the Structured Infrastructure for Multiformalism modeling and Testing of Heterogeneous formalisms and Extensions for SYStems (SIMTHESys) methodology [106], that allows users to create tools to solve arbitrary, user-defined classes of models. The key feature of the SIMTHESys approach, is the possibility to integrate different models, formalisms and solvers within a single framework, through the use of *Behavioural Facilities* (BFs). A Behavioural Facility can be seen as a translation layer between formalisms and solvers. BFs provide standard interfaces allowing to introduce new formalisms or new solvers simply adding a new implementation of those interfaces.

At the moment, SIMTHESysEr provides some predefined formalisms, such as some kind of queueing networks, Stochastic Petri Nets and Markov chains, along with some solution techniques for transient and steady state analysis. All the models that could be specified must have a finite state space. Although it does not provide a graphical interface, models can be designed using the Draw-Net software [86] and used by SIMTHESysEr.

The tool is written in Java, with some optional C++ modules, and the source code is freely available. Although installation instructions are provided only for Windows, it should run with little effort in every architecture that provides a Java Virtual Machine and a C++ compiler, including Linux and Mac OS X. The software and its documentation is in ongoing development.

### 4.3 A model example

In this section we give an example on how a model could arise from the analysis of a practical system, and we then show how such a model could be represented with the previously analysed tools.

Performance and reliability evaluation softwares should be used to model complex entities. Consider, for instance, a distributed system in which some services are shared among components, i.e., storage areas or computing facilities, while other are independent and can perform their work in parallel. Moreover, some tasks could be completed only if some other components have all finished some other jobs, i.e., there are some synchronisation points. Some of the activities in this system can be initiated by users, that exhibit a behaviour that could be dependent on their interaction with each others or with the system itself, while other ones could be of endogenous nature. In modelling the aforementioned system, one should identify

the indices to be evaluated, decide a set of sensible assumptions, and then choose among a variety of formalisms the most suitable ones to express those requirements. Describing this process is out of the scope of the present chapter, and thus we only note that a model of such complexity would be impractical to describe in this chapter. Therefore, we limit this section to a toy example, i.e., an ideal system with a reduced complexity and that exhibits some features which are typical of real systems and which are usually useful to model in practical applications.

Consider a system where  $N_u$  identical users, during a specific phase of a strict corporate workflow, request some computation to each of a set of  $N_s$  servers, which can perform their job in parallel. Each of these servers is, in turn, made of various software components, which use different hardware components, which can or cannot be shared among different servers. Once a user has collected all the information resulting from the computation of each server and he has spent some random time to merge them, he can continue to follow the workflow. A possible modelling strategy for such system is the following:

- Each of the users subject to the corporate workflow can be modelled by a process algebraic agent, e.g., a PEPA one, in which some activities have their own rates, while some other are synchronising with actions of other components, e.g., managers, clients, etc. Some action types represent requests to the servers, while others, with unknown rates, represent the completion of the entire set of tasks requested to the servers.
- The servers, along with their hardware and software components, can be modelled by a multiple-class and multiple-chain BCMP network, or, if more suitable, by a higher level hierarchical and/or modular formalism that can be translated in such networks, e.g., as described in [8].
- The synchronisation phase, in which the user has to wait for the results of all the computations of the servers, along the random time the user has to spend to collect those data, is best modelled by a very simple SPN.

Of those formalisms, only SPNs are directly supported by all the previously described tools. PEPA models are supported only by Möbius, while multiple-class and multiple-chain open BCMP networks can be reasonably well approximated using the multiple chain product form queueing networks formalism of SHARPE, which, however, cannot be used in composition with other formalisms. It is worth noting that an user with sufficient programming and theoretical skills could theoretically write a plugin to support the corresponding formalism in SIMTHESysEr.

Although model expressed in any of the aforementioned formalisms can be transformed into corresponding Continuous Time Markov Chains (CTMCs), thus allowing all the four tools to be able to solve the same set of problems, the translation process is difficult, and doing it manually reduces drastically the usefulness of an automated tool. It is sometimes possible to translate models among different high-level



formalisms, e.g. BCMP queueing networks in Stochastic Petri Networks [5]; however, those transformations often preserve only some of the characteristics of the original model, e.g., the average performance indices, while other aspects can vary. Moreover, doing those translations manually is, again, difficult or even unfeasible for larger models.

Once single submodels are conceived, one should combine them in a composed model. This is the most critical step in a multi-formalism framework, because of the aforementioned semantic issues and of the possible limitations of composition methods. For instance, while a hierarchical composition method is a clear and easy technique to combine models maintaining a layered abstraction, it is not always possible to identify a top-level model, with its associated formalism, which is able to express all the interaction among components. On the other hand, a flat composition method, i.e., one in which all the submodels are at the same level, could become unmanageable when the number of components increases.

While systems such as this example are quite common, none of the analysed software packages can represent directly the composition of the previous described submodels. However this is due to our choice of the formalism to be used to model the system itself. Other choices, although perhaps less immediate to understand, could have led to a model tractable by some or all the aforementioned tools.

## 4.4 Software Comparison

All the analysed software packages offer powerful tools for the design and analysis of complex systems, with particular regard to performance evaluation. All the software considered here arose in academic background, although some of them have subsequently become commercial products. From the point of view of a practitioner, while a fast and accurate solution for a vast set of models is desirable, the lack of implementation polishing or application support and maintenance could be a showstopper. On the other hand, academic users could be more interested in the availability of source code and in the extendability of the product.

Table 1 summarises the main features of all the multi-formalism softwares analysed, from which we can draw some conclusions, if not a relative ranking, among the analysed tools with respect to each of the represented qualitative characteristics.

- *Supported formalisms*: clearly SHARPE offers the widest choice of formalisms, and the possibility to have models that combine them. The only major family of formalisms not available in SHARPE is that of process algebras; however, these are supported by Möbius. While SIMTHESysEr can potentially support arbitrary formalisms through the use of pluggable components, at the moment their selection is quite restricted.
- *Solution techniques*: each of the software packages supports different solution methods for transient and steady state analysis, both analytical and through

simulations. While SIMTHESysEr could support any method available in plugins, at the moment this feature is not really exploited. SMART supports also model checking algorithms.

- *User interface*: only SHARPE and Möbius have graphical user interfaces. The one of Möbius has a little more modern design.
- *Documentation*: Möbius has, in our judgment, the best documentation. SHARPE and SMART are also very well documented. SIMTHESysEr, due to the fact that it is an ongoing project, has very little documentation.
- *License*: SIMTHESysEr is the only tool with a freely available source code. SHARPE and Möbius are proprietary softwares which allows academic users to get the software for free. The SMART license schema is on request, and, as stated before, at the time of the writing of this chapter we have not yet received an answer to our enquiry.
- *Maintenance and support*: SHARPE and Möbius are actively maintained and supported. The maintenance status of SMART is unclear; however, the documentation and the supported platforms (see below) suggests that the software was maintained at least until the recent past. SIMTHESysEr is part of an ongoing project, and thus it is still in development.
- *Platforms*: Möbius supports the three major operating systems for personal computing, i.e., Linux, MacOS X and Windows. The documentation of SMART also states the support of those three platforms. SHARPE, at least in solder version, supports Linux, Solaris and Windows, but the latest release seems to be packaged only for Windows. SIMTHESysEr should run for every platform that has Java support and a C++ compiler; however, its installation is documented only on Windows.

Each of the analysed softwares has peculiar characteristics which make them suitable for different purposes. As previously stated, SIMTHESysEr is the only product with a freely available source code, even if the license terms are not explicitly specified in the source itself. The aim of the project, more than building a finished design product, seems to be the definition of an infrastructure, based on the SIMTHESys methodology, to combine different design formalisms and solvers. As previously noted, the tool, while functional, is in a development stage, and thus cannot be easily used in a production environment. The other software analysed, i.e., SHARPE, Möbius and SMART are closed-source products. SHARPE is widely recognised as the earliest multi-formalism performance evaluation software still actively maintained. It offers a wide plethora of formalisms, and it is well documented both on the technical aspects by online references and on the design formalism aspects by a popular book on performance evaluation [170]. It also notable because it is the only tool to offer a comprehensive support for the most used classes of queuing

networks. Möbius is another mature product, with a considerable implementation polishing and a very detailed and updated documentation. While the set of supported formalism is less abundant than the one of SHARPE, the comprehensive user interface, the support for modern environment and the availability of instruments to analyse the results provided by the solvers may make this tool a better choice for practitioners who are not interested in the formalisms supported by SHARPE. SMART, is a very well documented product. However, it lacks a graphical user interface and supports a limited set of formalisms. The main strength of the tool lays on the availability of a programming language for the specification of models and on the support for model checking techniques. The lack of a public license scheme may be drawback for professional users.

Overall, while all these tools have remarkable qualities and are of great interest for academic users, from a practitioner point of view the most interesting products are Möbius, due to its implementation quality, and SHARPE, due to the wide variety of supported formalisms.

As a final remark, we point out the limited number of existing tools that allow for the solution of models expressed using different formalisms. As we noted in the introduction, the differences in the semantics of formalisms often poses significant difficulties in defining how they should integrate. Moreover, while efficient solution techniques may have been devised for some limited class of models, specified using a particular formalism, solution of general cases are often difficult and time and space consuming. Designing and building such tools requires a significant effort and a considerable amount of resources, which often cannot be afforded by single universities and research groups.

## 4.5 Conclusion

In this chapter we surveyed the state of the art of performance and reliability evaluation packages, with a particular focus on those that allows the modelling of systems using more than one class of formalisms. While all the surveyed tools are complex software packages with an impressive set of features, we noticed that there is still room for many improvements in this field. As noted before, combining different formalisms arises some difficulties in defining the semantics of their integration. Moreover solution algorithms defined only for a class of models expressed in a single formalism could not be applicable to composed models. In particular, while well-known product form solutions exist for queueing networks or stochastic Petri nets, models arisen from their combination could be difficult to solve efficiently for current tools. Since the advent of the RCAT theorem [89] and its extensions [91, 92], the presence of many intra-formalism product forms were proven [17, 21]. Iterative algorithms that solve models expressed as a set of cooperating PEPA agents, such as the tool described in Chapter 5, or that compute directly the reversed rates for known higher-level formalisms, could be used to efficiently solve such models. However,

	<b>SHARPE</b>	<b>Möbius</b>	<b>SMART</b>	<b>SIMTHESysEr</b>
<b>Supported Formalisms</b>	Markov and Semi-Markov, RBDs, FTs, RGs, MCPFQN, GSPNs, SPGs	SAN, Buckets and Balls, PEPA <sub>k</sub> , Fault Trees	Markov Chains, SPNs	pluggable, ATM Markov Chains, QNs (limited), SPNs
<b>Solution Methods</b>	Numerical (Approximate and Exact)	Exact Numerical, Simulation	Numerical	pluggable, ATM CTMC solution and simulation
<b>User Interface</b>	Textual, GUI	Textual, GUI	Textual	Textual
<b>Platforms</b>	Windows (Linux and Solaris in older versions)	Linux, Mac OS X, Windows	Linux, Mac OS X, Windows	Windows (other platforms may work)
<b>License</b>	Proprietary, no cost for academic users	Proprietary, no cost for academic users	Not Specified (on demand)	Source code freely available
<b>Maintenance</b>	Supported and updated	Supported and updated	Not recently updated	In active development

Table 4.1: Characteristics of multi-formalism software packages.

---

only a small subset of the possible stochastic models are in product form, thus those methods cannot be generally applied. Other difficulties arise in the restrictions in the ways an user can combine models, e.g., either only in a hierarchical fashion or in a flat one, thus reducing the expressivity of such compositions. In Chapter 9 we will present a tool that integrates SIMTHESysEr (Section 4.2) and the tool described in Chapter 5 in order to perform the analysis of some classes of multi-formalism product-form models.



# II

---

## Contributions





---

# 5

## Algorithmic Product form detection and solution

### 5.1 Introduction

This chapter describes a tool, initially presented in [15], that given the description of a set of cooperating CTMCs (i.e., when some transitions in one chain force transitions in other chains) it decides whether the model is in product-form and, in this case, computes its stationary distribution. The tool is based on the algorithm presented in [132], which is briefly summarised in Section 5.2. Since the analysis of the product-form is performed at the CTMC level, it is able to study product-form models that are originated from different formalisms, such as exponential queueing networks, G-networks or queueing networks with blocking. To this aim, we observe that it is important to decouple the analyser and the model specification interface (MSI). We propose both a Java implementation of the analyser and a general MSI. Note that multiple specification interfaces may be implemented according to the modeller needs. With this tool, a modeller has a library of product-form models that, even if they were created using some (possibly high-level) formalism, are stored as stochastic automata, basically a CTMC with labelled transitions allowing self-loops or multiple arcs between states. Using the MSI, which acts as a client with respect to the analyser, the various sub-models can be instantiated and their interactions be specified. The operations that the modeller performs in the MSI are translated into commands for the server side, i.e., the analyser. The analysis is requested from the MSI, computed by the analyser and displayed by the MSI. We have also developed a textual interface that will not be presented in this chapter to allow the usage of the analyser from non-graphical clients.

Moreover in Section 5.3 we describe an extension of the algorithm used by the tool, originally introduced in [9], which is able to opportunely truncate models with infinite state space. We show how the extended algorithm efficiently computes the solution of non-homogeneous models, e.g., models consisting of Jackson queues [107], G-queues with positive and negative customers [78], G-queues with catastrophes (or jumps back to zero) [50, 73].

## 5.2 The INAP Algorithm

This section briefly summarises the original formulation of INAP given in [132]. We refer to the original paper for the formal algorithm definition and considerations about its efficiency and convergence. Let us consider a set of cooperating models  $S_1, \dots, S_N$ , and let  $\pi_i^{(f)}$  be the steady-state distribution of model  $i = 1, \dots, N$  at the  $f$ -th algorithm iteration. We denote the reversed rate of cooperating label  $a$  at the  $f$ -th iteration by  $K_a^{(f)}$ . Finally, let  $\varepsilon$  be the precision required and  $T$  the maximum number of iterations. Let us assume that the structural RCAT conditions are satisfied, i.e., conditions 1 and 2 of Theorem 2.

The INAP algorithm, in its base version, operates as follows:

1. Initialisation:  $f \leftarrow 0$ , set up randomly  $\pi_i^{(0)}$  for all  $i = 1, \dots, N$
2. For all synchronising transitions  $a$ , compute  $K_a^{(f)}$  as the mean of the reversed rates of the transitions labelled by  $a$  using  $\pi_i^{(f)}$  with  $i$  such that  $a \in \mathcal{A}_i$
3. For all  $j = 1, \dots, N$ , set  $S_j^c = S_j \{a \leftarrow K_a^{(f)}\}$ , for all  $a \in \mathcal{P}_j$
4.  $f \leftarrow f + 1$
5. For all  $i = 1, \dots, N$  compute  $\pi_i^{(f)}$  as the stationary solution of  $S_i^c$
6. If there exists  $i \in [1, N]$  such that  $\pi_i^{(f)} \neq \pi_i^{(f-1)}$  within precision  $\varepsilon$  and  $f \leq T$  cycle to Step 2
7. Terminate with one of the following options:
  - If  $f > T$  return *No product-form solution found*
  - For all synchronising transitions  $a$ , use  $\pi_i^{(f)}$  with  $i$  such that  $a \in \mathcal{A}_i$  to check if the reversed rates of all transitions labelled by  $a$  are constant.
    - In case of positive check then return for all  $i = 1, \dots, N$  solution  $\pi_i^{(f)}$
    - In case of negative check then return *No product-form solution found*

Note that the steps of the algorithm are computable if the state spaces of models  $S_1, \dots, S_N$  are finite. However, in [132] it is shown that thanks to the special structure of the stations in G-networks with negative customers and in Jackson networks, these can be modelled with just a pair of states. In this way, the algorithm has been proved to be equivalent to the Jacobi iterative scheme on the traffic equation system in case of Jackson networks, and to Gelenbe's iterative scheme in case of G-networks [78]. On the other hand, the special class of models with infinite state space for which this technique may be adopted is very small, i.e., those whose underlying process is a Birth and Death process. Most of the models with infinite state space cannot be reduced to this analysis and a different algorithm has to be defined, as we shall introduce in the next section. The following examples illustrate two well-known models with infinite state space that cannot be reduced in such a way.

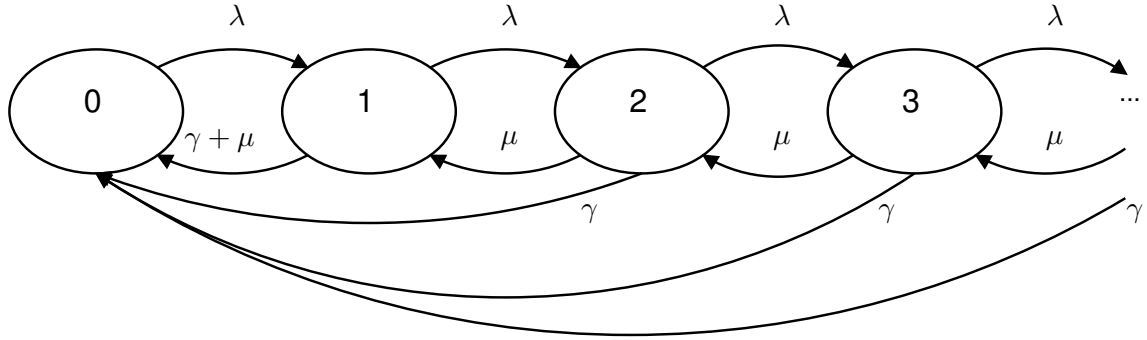


Figure 5.1: CTMC underlying a G-queue with catastrophes.

**Example 2** (G-queues with catastrophes). *Consider a queueing system where customers and triggers arrive according to independent Poisson processes with rates  $\lambda$  and  $\gamma$ , respectively. The service time is exponentially distributed with rate  $\mu$  and at a trigger arrival epoch all the customers in the queue are removed, i.e., the state jumps back to zero. The product-form properties of this model are studied in [50, 73]. Figure 5.1 depicts the stochastic process underlying this model. The G-queue with catastrophes has a geometric steady-state distribution [50]  $\pi(k) = \pi(0)\rho^k$  with*

$$\rho = \frac{\lambda + \gamma + \mu - \sqrt{\lambda^2 + \gamma^2 + \mu^2 + 2\lambda\gamma + 2\mu\gamma - 2\lambda\mu}}{2\mu}. \quad (5.1)$$

This model cannot be reduced to one with finite state space using the technique described in [132] because the process depicted by Figure 5.1 is not a Birth and Death one. Note that the reversed rates of the departure transitions (i.e., those going from state  $i+$  to state  $i$ ,  $i \geq 0$ , with rate  $\mu$ ) are constant and equal to  $\pi(k+1)/\pi(k)\mu = \rho\mu$ .

**Example 3** (Multiclass FCFS queue with single exponential server). *Consider a queue with  $R$  classes of customers, First Come First Served (FCFS) discipline, single exponential server. Let  $\lambda^{(r)}$  be the rate of the independent Poisson process modelling the arrivals of class  $r$  customers, and  $\mu^{(r)}$  be the service rate for the same class,  $1 \leq r \leq R$ . The model has been widely studied in literature, and the product-form properties are derived in [26]. The state is a finite and unlimited vector whose dimension is equal to the number of customers in the queue at a given time. Its  $i$ -th component is the class of the  $i$ -th oldest customer in the queue. The oldest customer is then in position 1 and is the customer being served. At a job completion event the vector length is reduced by a unit and all customers move ahead in the queue. It is well-known that the condition for product-form is that  $\mu^{(1)} = \mu^{(2)} = \dots = \mu^{(r)} = \mu$ . As for the previous example, the technique adopted in [132] cannot be applied since the CTMC is not a Birth and Death process.*

### 5.3 INAP for models with infinite state spaces

In this section we describe the new algorithm for the analysis of product-form models. In general, when one models a system with components whose state spaces may be infinite, the problem is to provide a suitable truncation of the process. We first introduce the algorithm input, which is slightly different from that of original INAP, then we describe the main idea, and finally we formalize the algorithm definition. We call this improved algorithm INAP+.

#### 5.3.1 The algorithm input

INAP input consists of sub-model descriptions in form of square matrices  $L_k \times L_k$ , where  $L_k$  is the number of states of  $S_k$ ,  $1 \leq k \leq N$ . These matrices describe the transitions among the states and the associated labels. Another input is the set of the synchronising labels. INAP+ works with models with infinite state spaces, hence the matrix-form description presented for INAP in [15] is no more usable.

Models are described through a program which allows the definition of infinite state spaces, for instance a parametrised PEPA language (without the synchronising operator). Examples of infinite state space models described in this way are quite common, e.g., they can be found in [89]. Finally, we need to define an operator to reduce infinite state spaces to finite ones. Note that, if an automaton is closed, i.e., it has not passive transitions, the truncation may be done before the algorithm runs. This is what usually happens when applying numerical techniques for the analysis of open models. However, in this context, we want the user to be able to specify how to truncate the process in a parametric way, i.e., referring to values  $K_a$  for each  $a \in \mathcal{P}_i$ . The following example illustrates this idea for a trivial case.

**Example 4** (Truncation of a Jackson queue). *Consider a queueing station in a Jackson network [107]. A suitable model for this queue is depicted by Figure 5.2. Since the rates of transitions labelled by  $a$  are not known we are not able to decide*

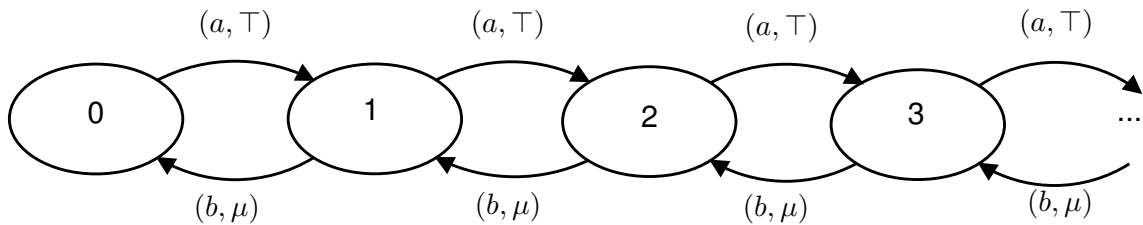


Figure 5.2: Example of CTMC underlying a Jackson queueing station.

which states have a stationary probability higher than a given  $\tau \in \mathbb{R}^+$ . However, if RCAT is applicable, we know that all those passive transitions have the same rate  $K_a$ . At each iteration of INAP, a value for the passive transitions is computed,

hence, at each iteration we can compute a (possibly different) truncation. In this example, a possible choice is to truncate the process for  $n > \ln \tau / (\ln(K_a) - \ln(\mu))$ . This means that the parametric truncation is useful when the station is embedded in a heterogeneous queueing network (i.e., not a Jackson network whose analytic solution is known) and hence the rate of the passive transition cannot be easily derived.

Formally, we define an operator  $\mathcal{R}^\tau$ ,  $\tau \in \mathbb{R}^+$  on a stochastic automaton  $S$ , with the following properties:

**Definition 2** (Truncation operator properties).

- $\mathcal{R}^\tau(S)$  has a finite number of states
- If  $m \in S$  does not belong to the state space of  $\mathcal{R}^\tau(S)$  then  $\pi(m) < \tau$
- Transitions in  $\mathcal{R}^\tau(S)$  are a subset of those in  $S$ .  $n \xrightarrow{a} n'$  is both in  $S$  and  $\mathcal{R}^\tau(S)$  if and only if  $n, n'$  belong to  $\mathcal{R}^\tau(S)$
- $\mathcal{R}^\tau(S)$  is ergodic
- $\mathcal{R}^\tau(S)$  has at least one transition labelled by  $a$ , for each label  $a$  involved in the synchronisation

Note that these requirements are not strict and this type of truncation is often done when one wants to analyse an open model by means of the GBE solution and knows all the rates of the underlying chain. However, when defining an interaction among several components it is not obvious at all where the truncation should be applied, because some model parameters are unknown. Finally, it is worthwhile to point out that, as expected, in order to satisfy Definition 2 requirements, it is sufficient to provide an upper bound for the steady-state solution of the model, and not its exact expression. For what concerns the notation, the algorithm takes as input the set of models  $S_1, \dots, S_N$ , and the associated operators  $\mathcal{R}_1^\tau, \dots, \mathcal{R}_N^\tau$ . How to simply express the  $\mathcal{R}^\tau$  operators is outside the scope of this work, but in the application framework we are developing it is embedded in the model definition, where each transition may be labelled with a condition, which in general depends on  $\tau$  and other transition rates. A transition and its incoming state belong to  $\mathcal{R}^\tau(S)$  only if its associated condition is satisfied.

### 5.3.2 Main idea of the algorithm

The algorithm presented in Section 5.2 is modified in two ways. The first one is how the algorithm manages the dynamic truncation of the sub-model state spaces, and the second one is how the reversed rates of the active labels are computed at each iteration.

**Dynamic truncation of the state spaces.** At each iteration  $f$ , values  $K_a^{(f-1)}$ , i.e., the reversed rates computed in the previous step, are used to close the automata and to truncate them according to operator  $\mathcal{R}_i^r$ ,  $1 \leq i \leq N$ . This technique is based on the observation that the knowledge at step  $f$  of the temporary reversed rates of the transitions  $a \in \mathcal{P}_i$  associated with automata  $S_i$ ,  $1 \leq i \leq N$ , completes the information needed to compute the result of the application of operator  $\mathcal{R}_i^r(S)$ .

**Iterative parametrisation of the sub-models.** The computation of the reversed rates is changed with respect to what was proposed in INAP, and is not based on the computation of the means of the reversed rates of all the active transitions. Roughly speaking, we would like the reversed rates of the active transitions incoming into states with low probability to count less in the computation of  $K_a^{(f)}$  than those incoming into states with higher stationary probabilities. The reason for this can be understood through an analysis of RCAT proof in [89] where the author points out the relation of product-form solutions with the reversed processes. Indeed, when considering the reversed process of a closed agent  $S_i^c$  all the active transitions with the same label have the same rates and are outgoing from every state of the model. This means that if  $n$  is a state with a very low stationary probability, an active transition outgoing from  $n$  has a lower impact on the model behaviour than one outgoing from a state with higher stationary probability. For this reason we apply a weighted sum approach, i.e.:

$$K_a^{(f)} = \sum_{\substack{n, n' \in S_i: \\ \exists n \xrightarrow{a} n'}} \bar{q}_i(n \xrightarrow{a} n') \pi_i^{(f)}(n') = \sum_{\substack{n, n' \in S_i: \\ \exists n \xrightarrow{a} n'}} \pi_i^{(f)}(n) q_i(n \xrightarrow{a} n') \quad (5.2)$$

Since each state, by hypothesis, has exactly one incoming active transition, Equation (5.2) gives the weighted mean of the reversed rates. Note that Equation (5.2) computes  $K_a^{(f)}$  using all the arcs labelled by  $a$  outgoing from the states regardless to the fact  $n'$  belongs to  $\mathcal{R}_i^r(S_i^c)$  or not. This is possible because the expression of  $K_a^{(f)}$  is independent of the computation of  $\pi_i^{(f)}(n')$ .

### 5.3.3 Formal definition of INAP+

We shall now formalise the INAP+ algorithm. We use  $\boxtimes_{k=1}^N S_k$  to denote the joint model, where  $S_k$  is a cooperating automaton and  $\mathcal{L} = \bigcup_{k=1}^N \mathcal{A}_k = \bigcup_{k=1}^N \mathcal{P}_k$  is the set of cooperating labels. Note that  $\pi_k^{(f)}$ , with  $1 \leq k \leq N$ , can be seen as a vector, but its size depends on  $f$ . When comparing  $\pi_k^{(f)}$  with  $\pi_k^{(f-1)}$  we consider them different if their size are not equal, while in case they are equal their components in the same position must not differ more than  $\varepsilon$ . Note that although we keep track of all the stationary probability vectors  $\pi_k^{(f)}$  for readability, only two of them are actually necessary, i.e., the current,  $f$ , and the previous ones,  $f - 1$ . Algorithm 1 shows the formal definition.

---

**Algorithm 1:** Simplified algorithm.

**Input:** agents  $S_1, \dots, S_N$  and their truncation operators  $\mathcal{R}_i^T$ ; precisions  $\varepsilon, \tau$ ;  
maximum number of iterations  $T$

**Output:** unnormalized stationary distribution  $\boldsymbol{\pi}$  of  $\bigotimes_{k=1}^N S_k$   
 $\mathcal{L}$

Set up initial sizes of  $S_i$  for all  $k = 1, \dots, N$

$f \leftarrow 0$  Randomly initialize  $\pi_k^{(f)}$  for all  $k = 1, \dots, N$

**repeat**

$f \leftarrow f + 1$

    /\* Reversed rates \*/

**for**  $k = 1, \dots, N$  **do**

**foreach**  $a \in \mathcal{A}_k$  **do**

            ⊥ Compute  $K_a^{(f)}$  using Equation (5.2)

    /\* Prepare the models for the new iteration \*/

**for**  $k = 1, \dots, N$  **do**

$S_k^R \leftarrow \mathcal{R}_k^T(S_k \{\forall a \in \mathcal{P}_k, a \leftarrow K_a^{(f)}\})$

        ⊥ Compute  $\pi_k^{(f)}$  as the solution of  $S_k^R$

**until**  $f > T$  **or**  $\forall k = 1, \dots, N. \pi_k^{(f)} =_\varepsilon \pi_k^{(f-1)}$ ;

/\* Check if a fixed point has been reached \*/

**if**  $f > T$  **then**

    ⊥ Output *No product-form identified*

**else**

    /\* Check if the reversed rates are constant \*/

$ans \leftarrow \mathbf{true}$

**for**  $k = 1, \dots, N$  **do**

**foreach**  $a \in \mathcal{A}_k$  **do**

$\Lambda \leftarrow \{\pi_k^{(f)}(n)/\pi_k^{(f)}(n')q_k(n \xrightarrow{a} n') : n \xrightarrow{a} n' \in S_k^R\}$

**if**  $\max \Lambda - \min \Lambda > \varepsilon$  **then**

                ⊥  $ans \leftarrow \mathbf{false}$

**if**  $ans$  **then**

        ⊥ Output  $\pi_k^{(f)}$  for  $k = 1, \dots, N$

**else**

        ⊥ Output *No product-form identified*

---

### 5.3.4 Convergence, termination, complexity and optimisations

INAP+ shares with its predecessor the absence of a proof of convergence for general cases, although positive results for Jackson queueing networks and G-networks with negative customers are provided in [132]. In our tests we have not found an example of a false negative, i.e., a model that is known to be in product-form but for which the algorithm could not find a solution. Note that special cases in which the basilar iterative schemes on the traffic equations do not work as that presented in [74] cannot be modelled in our framework because they involve a trigger definition that is not pairwise. The termination of the algorithm is ensured by the introduction of a maximum number of iteration. This is needed because although the algorithm cannot diverge, it may exhibit an undesired cyclic behaviour. The complexity of the algorithm depends on the state space cardinalities of subsystems. Since these are dynamically computed we cannot easily predict the complexity for general cases. However, if at each step  $f$ , each of the  $N$  agents has  $r$  states, then the complexity of the iteration is  $\mathcal{O}(Nr^3)$ .

The optimisations proposed in [132] are still applicable to INAP+. We just summarise them:

- Active transition self-loops: since the reversed rate of a self-loop is equal to its forward rate, if two self-loops with the same label  $a$  have different rates, the product-form does not exist. If they all have the same rate, that rate must be the reversed rate  $K_a$ .
- Sub-models can be solved in a suitable order to reduce the algorithm complexity. This can be done by an analysis of the strong connected components in the graph of dependencies among the sub-models, where  $S_i$  depends on  $S_j$  if there exists a label  $a \in \mathcal{P}_i \cap \mathcal{A}_j$ .
- The solution of each sub-model can be computed by independent computational units. This improves the algorithm efficiency by exploiting parallelism.

## 5.4 Model and cooperation encoding

In this section we explain how to encode a problem solvable by our algorithm in a representation suitable to be used by a computer software. Moreover, we show how to enhance reusability and modularity of models through an extension of cooperation semantics.

Let us suppose to have  $N$  model  $S_1, \dots, S_N$  that cooperate as specified in Chapter 3, i.e., some transitions in a model  $S_i$  force other transitions in a model  $S_j$ ,  $i \neq j$ . At a low-level we can describe each model by a set of labelled matrices:  $\mathbf{M}_i^a$  is the matrix with label  $a$  associated with model  $S_i$ . Labels may be chosen arbitrarily



when a model is defined. However, we always assume that every model has at least one label called  $\epsilon$ . We consider, at first, models with a finite number of states,  $Z_i$ .  $\mathbf{M}_i^a$  is a  $Z_i \times Z_i$  matrix with non-negative elements that represent the transition rates between two states of the model. Note that self-loops, i.e., transitions from a state to itself, are allowed. The infinitesimal generator  $\mathbf{Q}_i$  can be easily computed as the sum of all the matrices associated with a model, where the diagonal elements are replaced with the opposite of the sum of the extra-diagonal row elements. If the stationary distribution  $\boldsymbol{\pi}$  exists (and hereafter we will work under this hypothesis) then it can be computed as the unique solution of  $\boldsymbol{\pi}\mathbf{Q} = \mathbf{0}$  subject to  $\boldsymbol{\pi}\mathbf{1} = 1$ . From  $\boldsymbol{\pi}$  we can compute the rates in the reversed process associated with each label using Equation (3.1). Suppose that  $\mathbf{M}_i^a[\alpha, \beta] > 0$ , with  $1 \leq \alpha, \beta \leq Z_i$  and  $1 \leq i \leq N$ , then the reversed rate of this transition, denoted by  $\overline{\mathbf{M}_i^a[\alpha, \beta]}$  is defined as follows:

$$\overline{M_i^a[\alpha, \beta]} = \frac{\pi(\alpha)}{\pi(\beta)} M_i^a[\alpha, \beta]. \quad (5.3)$$

Let us show how we specify the interaction of two models. According to RCAT restrictions, we just deal with pairwise interactions, i.e., a transition in a model may cause a transition for at most another model. The cooperation semantics used in this chapter (but also in [89]) is very similar to that specified by PEPA (see Section 2.3.3). Consider sub-models  $S_i$  and  $S_j$  and suppose that we desire to express the fact that a transition labelled with  $a$  in  $S_i$  can occur only if  $S_j$  performs a transition labelled with  $b$ , and vice-versa. Specifically, if  $S_i$  and  $S_j$  are in states  $s_i, s_j$  such that they are able to perform a transition labelled with  $a$  and  $b$ , respectively, that take the sub-models to state  $s'_i$  and  $s'_j$ , then they can move simultaneously to state  $s'_i$  and  $s'_j$ . The rate at which this joint transition occurs is decided by the active sub-model that can be  $S_i$  or  $S_j$ . We express such a cooperation between  $S_i$  and  $S_j$ , with  $S_i$  active, as follows:

$$S_i \underset{(a^+, b^-)}{\overset{y}{\times}} S_j,$$

which means that transitions labelled by  $a$  in  $S_i$  are active with respect to the cooperation with transitions labelled by  $b$  of  $S_j$  and that, in the resulting model, the joint transitions are labelled with  $y$ . The fact that the resulting model is still Markovian should be obvious because the synchronisation inherits the properties derived for that of PEPA. Note that the major difference is that we can synchronise different labels and assign a different name to the resulting transitions. This happens because we would like a modeller to be able to use a library of models whose labels have a local scope. In this way the library items can be created independently and instantiated several times in the same model.

**Example 5** (Example of cooperation). *Suppose we would like to model the trivial tandem queueing network depicted in Figure 5.3 with two identical exponential queues with finite capacities  $B$ . When the first queue is saturated, arrivals are lost.*

When the second queue is saturated at a job completion of the first queue, the customer is served again (repetitive service blocking). Customers arrive to the first queue according to a Poisson process with rate  $\lambda$ . A queue can be described by three

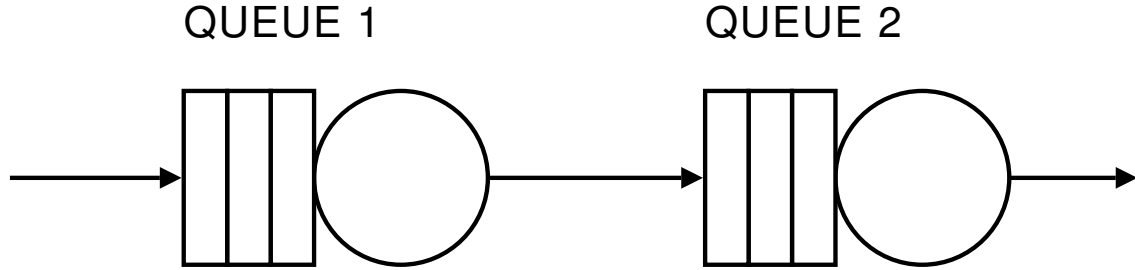


Figure 5.3: Tandem of two exponential finite capacity queues.

matrices with dimension  $B \times B$ :

- $\mathbf{M}^e = \mathbf{0}$  that describes the transitions that cannot synchronise (something like the private part of the model).
- $\mathbf{M}^a$  where  $\mathbf{M}^a[\alpha, \beta] = \lambda$  if  $\beta = \alpha + 1$  or  $\mathbf{M}^a[\alpha, \beta] = 0$ , otherwise. This matrix describes the transitions corresponding to arrival events.
- $\mathbf{M}^d$ , where  $\mathbf{M}^d[\alpha, \beta] = \mu$  if  $\beta = \alpha - 1$  or  $\mathbf{M}^d[\alpha, \beta] = 0$ , otherwise. This matrix describes the transitions corresponding to job completion events.

Consider two instances of this model,  $S_1$  and  $S_2$ . The tandem network of Figure 5.3 can be described by the model  $S_1 \times_{(d^+, a^-)}^y S_2$ .

A pairwise cooperation may involve more than one label. In this case we may write:

$$S_1 \begin{matrix} y_1 \\ \times \\ (a_1^+, b_1^-) \end{matrix} \begin{matrix} y_2 \\ \times \\ (a_2^-, b_2^+) \end{matrix} S_2$$

to specify that  $S_1$  ( $S_2$ ) is active on  $y_1$  ( $y_2$ ) and passive on  $y_2$  ( $y_1$ ) with transitions labelled  $a_1$  ( $b_1$ ) and  $a_2$  ( $b_2$ ), respectively.

The following operator allows us to change all the rates of a matrix labelled by  $a$ :  $S_1 \{a \leftarrow \lambda\}$  is the sub-model  $S_1$  with only matrix  $\mathbf{M}^a$  modified so that all its non-zero elements are set to  $\lambda$ .

## 5.5 Tool

In this section we describe some salient characteristics of the proposed tool. First, we explain our approach in the specification of the interactions between the sub-models. Then, we describe the client-server architecture and illustrate its strengths.

### 5.5.1 Specifying the interactions

Let us consider again the model depicted by Figure 5.3 with a variation, i.e., after a job completion at the first station the customer may exit the system with probability  $p$  or go to the second station with probability  $1 - p$ , as depicted by Figure 5.4. We

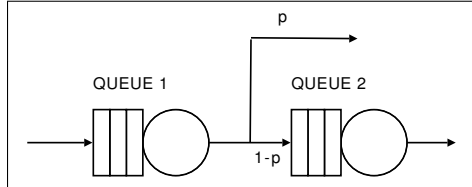


Figure 5.4: Example of Figure 5.3 model with variated probabilistic routing.

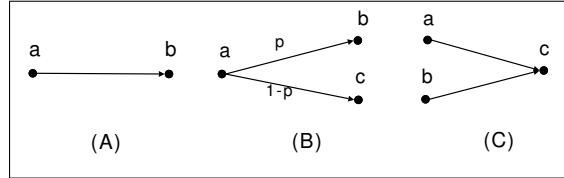


Figure 5.5: Types of connections between labels.

note that the CTMCs underlying the first and second queue are different, and we could not use two instances of the same model anymore. Indeed, in the first queue the transition corresponding to a job completion from state  $j$  to state  $j - 1$  must be split in two: one synchronising with the arrivals in the second queue with rate  $(1 - p)\mu_1$  and one without synchronisation with rate  $p\mu_1$ . We decided that this splitting of transitions should be done automatically by our tool, so that the library of sub-models can be defined without any knowledge about the future usage and connections.

From the modeller point of view, a sub-model is seen just as a black box where the labels are exported, i.e., a model specification consists of a set of connections among instances of modules. The simplest possible connection between two labels is that depicted by Figure 5.5-(A). Note that we use a graphical representation of the connections which is coherent with the proposed MSI we developed, however different approaches are possible (such as a PEPA-like syntax). Figure 5.5-(A) illustrates a label  $a$  of a sub-model that interacts with a label  $b$  of another sub-model. The arrow is oriented, meaning that  $a$  is active and  $b$  is passive. This specification of synchronisation does not require any modification to the structure of the active or passive sub-models. Let us now consider Figure 5.5-(B). In this case the active action  $a$  of one sub-model synchronises with passive actions  $b$  (with probability  $p$ ) or  $c$  (with probability  $1 - p$ ) of other sub-models. In this case, we need to alter the structure of the active model. Recall that matrix  $\mathbf{M}^a$  represents the transitions labelled by  $a$ . Then we define  $\mathbf{M}^{a'} = p\mathbf{M}^a$  and  $\mathbf{M}^{a''} = (1 - p)\mathbf{M}^a$ . Hence, in the active sub-model, matrices  $\mathbf{M}^{a'}$  and  $\mathbf{M}^{a''}$  replace matrix  $\mathbf{M}^a$ . Note that this technique can be applied to an arbitrary number of probabilistic synchronisations under the obvious constraint that the synchronisation probabilities must sum to a value that is less or equal to 1. Suppose that the sum of the probabilities  $p_1, \dots, p_K$  is  $p_t < 1$  (see Figure 5.4 for an example). In this case we have  $\mathbf{M}^{a^k} = p_k\mathbf{M}^a$  for  $k = 1, \dots, K$ , and  $\mathbf{M}^\epsilon$  (which is always present in a model description and represents

the transition that cannot synchronise) is replaced by  $\mathbf{M}^c + \mathbf{M}^a(1 - p_t)$ . We use the notation  $S_1 \times_{(a^+, b^-)}^{y, p} S_2$  to denote that  $a$  in  $S_1$  is active in the synchronisation with  $b$  in  $S_2$ , and the synchronisation is called  $y$  and occurs with probability  $p$ . The latter case is depicted by Figure 5.5-(C) where two active labels  $a$  and  $b$  (that can belong to the same or different sub-models) synchronise with the same passive label  $c$ . In this case we simply replace matrix  $\mathbf{M}^c$  of the passive model with two matrices  $\mathbf{M}^{c'}$  and  $\mathbf{M}^{c''}$  identical to the former (we do not need to modify the rates since they are replaced with the rate of the corresponding active transitions).

**Example 6** (Application to the model of Figure 5.4). *Let us show how we model the tandem of exponential queues with finite capacities  $B$  depicted by Figure 5.4. We still consider two identical instances of the same sub-model which is described in Example 5. The user specifies in some way the interactions. The model corresponding to the second queue does not change, while that corresponding to the first queue becomes the following:*

- $\mathbf{M}^e = p\mathbf{M}^d$  that describes the transitions that cannot synchronise
- $\mathbf{M}^a$ ,
- $\mathbf{M}^{d'} = (1 - p)\mathbf{M}^d$ ,

where  $\mathbf{M}^a$  and  $\mathbf{M}^d$  are the matrices defined in Example 5.

We point out some notes about this approach to the specification of the sub-model interactions: 1) Its scope is to allow the specification of a model in spite of the synchronisations it will be involved in. For instance, if we have a model of a simple exponential queue, we can straightforwardly define a Jackson queueing network with probabilistic routing by simply instantiating several copies of the same model. Moreover, connections have a simple and intuitive meaning. 2) When an active label is split the infinitesimal generator of the sub-model does not change, i.e., its stationary distribution does not change. Moreover, if the reversed rates of the transitions corresponding to active label  $a$  are constant in the original model, then also the transitions corresponding to a split label associated with  $a$  have constant reversed rates. 3) The effects of the replication of passive label matrices on the algorithmic analysis of the product-form is that the rate associated with the passive transition is the sum of the (constant) reversed rates of every associated active transition. 4) Specifying pairwise interactions where the same label is simultaneously active and passive with respect to two or more synchronisations is not allowed. This characteristic is inherited from the semantics of the cooperation given in the theoretical paper which this tool is based on.

### 5.5.2 Client-server architecture

The tool consists of two parts: the analyser (the server) and the MSI (the client). The idea is that although we propose a graphical client side that exploits the

strengths of our modular approach and the specification of the module synchronisation, one could write one's own MSI in order to make it compatible with the favourite formalism.

The server opens an independent session for each MSI connected. It provides a protocol which is used by the MSI to: 1) Create/Import a sub-model, 2) Specify a synchronisation between two labels of two sub-models, 3) Require the solution of a model given a precision and a maximum number of iterations. In the first and second case the server just answers the client if the required operation has been executed correctly, while the latter one returns the following data: 1) A flag that specifies if the product-form has been identified, 2) The steady-state probabilities of each sub-model, 3) The reversed rates of all the active transitions. Note that knowing the reversed rates of the active transitions means knowing the solution of the system of traffic equations.

### 5.5.3 Use cases

In this section we illustrate some examples of case studies. We give a description of the model which is independent of the MSI that we adopt. We just focus on two well-known results about product-form, although several other examples may be easily produced.

**Jackson networks.** Jackson networks are easy to study because they are characterised by a linear system of traffic equations. However, in our framework, they require some attention since each sub-model (i.e., each exponential queue) has an infinite state space. As stated in Section 5.2, in this case we can simply represent the sub-model using just a pair of adjacent states. Whenever we apply this technique to reduce the infinite state space of a sub-model we must disable the RCAT structural check (Condition 1) because some transitions that are present in the real model are omitted in the finite one. Figure 5.6 shows the truncation of an exponential queue. If the synchronisations it will be involved in impose  $a$  to be passive and  $d$  to be active, we note that Condition 1 of RCAT is satisfied for the infinite model but is not satisfied for the reduced one (e.g., state  $n + 1$  does not have any incoming active transition or outgoing passive transition). Nevertheless, the algorithm may still be applied.

**Example 7** (Jackson network). *Consider the Jackson network depicted in Figure 5.7. A sub-model of an exponential queue consists of three matrices (states are in the order  $n$  and  $n + 1$ ):*

$$\mathbf{M}^\epsilon = \mathbf{0} \quad \mathbf{M}^a = \begin{bmatrix} 0 & \lambda \\ 0 & 0 \end{bmatrix} \quad \mathbf{M}^d = \begin{bmatrix} 0 & 0 \\ \mu & 0 \end{bmatrix}$$

*We also use a single-state sub-model to represent the external Poisson arrivals with  $\mathbf{M}^\epsilon = \mathbf{0}$  and  $\mathbf{M}^a = [\lambda]$ . Let  $\mu_i$  denote the service rate of Queue  $i$ ,  $1 \leq i \leq 3$ , and  $S$*

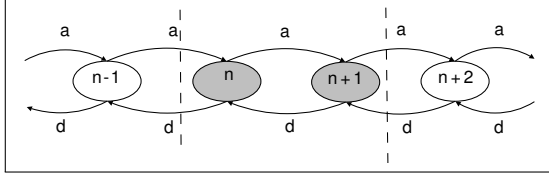


Figure 5.6: Truncation of the birth and death process underlying an exponential queue.

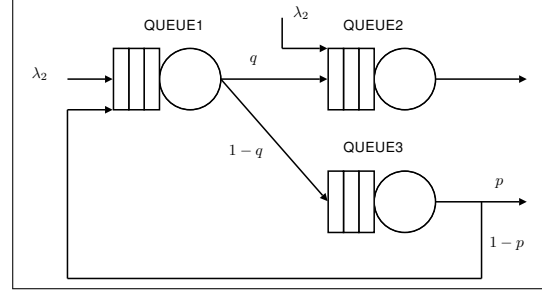


Figure 5.7: Jackson network of Example 7.

the library model for any queue and  $A$  for the external arrivals. Then we can write:

$$S_i = S\{d \leftarrow \mu_i\} \quad i = 1, 2, 3 \quad A_t = A\{a \leftarrow \lambda_1 + \lambda_2\}$$

The synchronisations are specified with the following commands to the server:

$$A_t \begin{matrix} y_1, \lambda_1 / (\lambda_1 + \lambda_2) \\ \times \\ (a^+, a^-) \end{matrix} S_1, \quad A_t \begin{matrix} y_2, \lambda_2 / (\lambda_1 + \lambda_2) \\ \times \\ (a^+, a^-) \end{matrix} S_2, \quad S_1 \begin{matrix} y_3, q \\ \times \\ (d^+, a^-) \end{matrix} S_2, \quad S_1 \begin{matrix} y_4, 1-q \\ \times \\ (d^+, a^-) \end{matrix} S_2, \quad S_2 \begin{matrix} y_5, 1-p \\ \times \\ (a^-, d^+) \end{matrix} S_3.$$

**G-networks.** We can model G-networks in our framework similarly as for Jackson networks. Note that although the models are different both in specification and in analysis, our tool treats them uniformly by exploiting the RCAT theoretical result. The truncation mechanism presented for Jackson queueing centers applies also to G-queues which consist of four matrices:  $\mathbf{M}^\epsilon$ ,  $\mathbf{M}^A$  representing the transitions for positive customer arrivals,  $\mathbf{M}^d$  representing those for job completion and  $\mathbf{M}^a$  for negative customer arrivals:

$$\mathbf{M}^\epsilon = \mathbf{0}, \quad \mathbf{M}^A = \begin{bmatrix} 0 & \lambda_A \\ 0 & 0 \end{bmatrix}, \quad \mathbf{M}^d = \begin{bmatrix} 0 & 0 \\ \mu & 0 \end{bmatrix}, \quad \mathbf{M}^a = \begin{bmatrix} 0 & 0 \\ \lambda_a & 0 \end{bmatrix}.$$

Although in the previous examples we have focused on infinite-capacity queueing systems, finite-capacity ones, such as Akyildiz's product-form queueing networks with blocking [1], have a finite state space, so the truncation mechanism is not needed [15].

### 5.5.4 MSI implementation example

In this Section we illustrate a possible implementation of the MSI. Recall that the tool client-server architecture allows for different MSIs according to the modeller's needs. We show a general-purpose MSI that is independent of the formalism used by the modeller. As an example we model the Jackson network depicted by Figure 5.7. Each sub-model is represented by a circle and arcs represent the synchronisations.

Each object, circle or arc, has a name. In the former case it is the sub-model name, in the latter it has the form  $y(a, b)$  that stands for  $S_1 \times_{(a+, b-)} S_2$ , where  $S_1$  is the sub-model from which the arc outgoes from, and  $S_2$  is the destination sub-model (see the screenshot in Figure 5.8). Clicking on a sub-model circle, a window appears

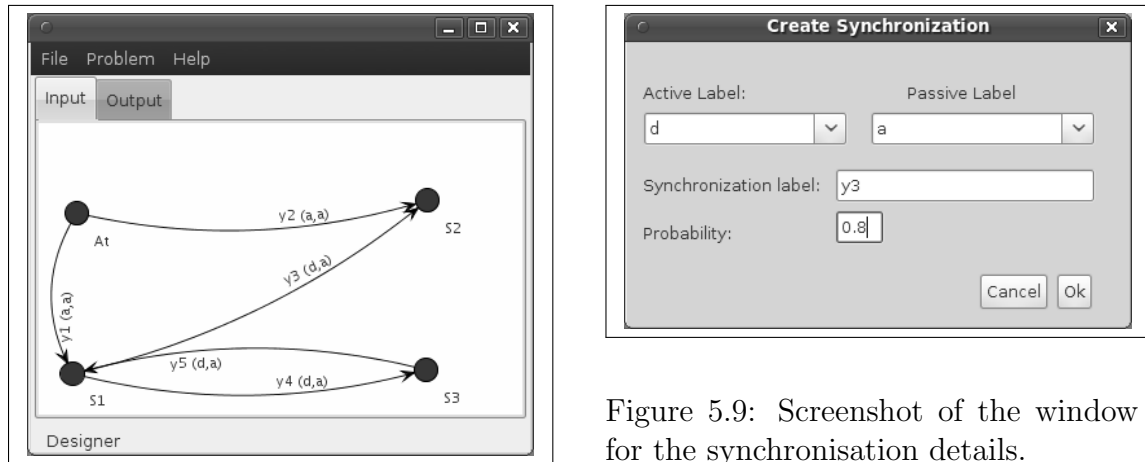


Figure 5.9: Screenshot of the window for the synchronisation details.

Figure 5.8: Screenshot of the model corresponding to the Jackson network of Figure 5.7.

with its description in matrix-form and the user is allowed to perform changes (add or remove transitions or change rates). When an arc is set between two sub-models, the window shown in Figure 5.9 appears. Note that, although one could point out that a standard tool for the analysis of Jackson networks may present a more intuitive interface, we would like to remark that this is the same interface we would use for any stochastic model that can be solved by the algorithm presented in [132]. However, one could also extend the MSI in order to associate a specific symbol to some sub-models of the library, but this is out of the scope of this presentation.

## 5.6 A Numerical Example

In this section we show how Algorithm INAP+ presented in Section 5.3 can be used to solve a heterogeneous model consisting of queueing station of different types.

Consider the example in Figure 5.10, a queueing network with 10 stations,  $S_1, \dots, S_{10}$ , where queue marked with letter J, G and C are Jackson queues, G-queues with negative customers and G-queues with catastrophes, respectively. Customers arrive from the outside to queue 1 according to a Poisson process with rate  $\lambda$ . Customers can move among queues or leave the system according to routing matrices  $\mathbf{R}^+ = [r_{i,j}^+]$  and  $\mathbf{R}^- = [r_{i,j}^-]$  for positive customers and triggers, respectively. For instance  $r_{i,j}^+$  denotes the probability for a customer to enter station  $j$  after a job

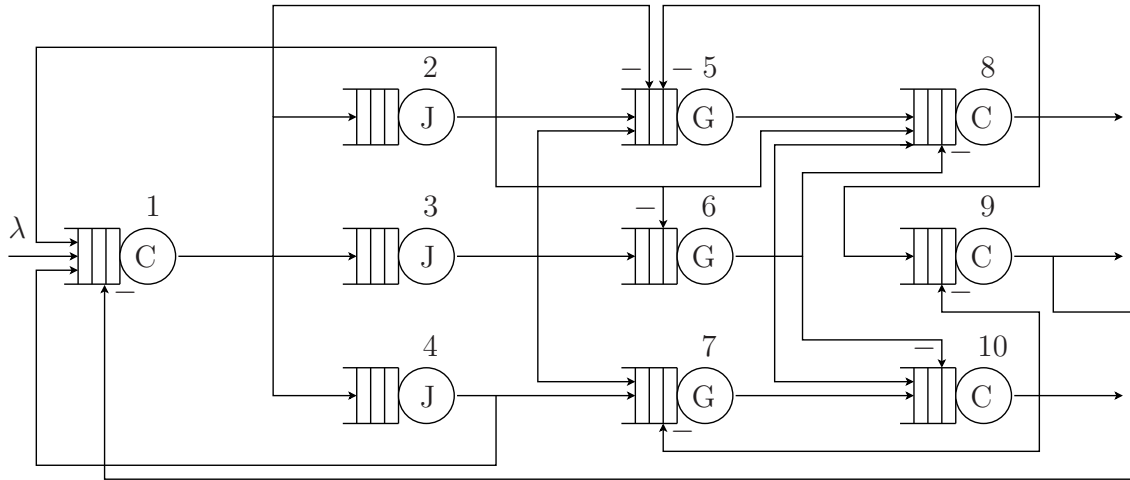


Figure 5.10: Example of a heterogeneous queueing network.

completion at station  $i$  as positive customer, while  $r_{i,j}^-$  denotes the probability for a customer leaving station  $i$  to join station  $j$  as a trigger. The effect of triggers depends on the type of the target station: if it is marked with G then the trigger removes one customer, while if it is marked with C, the trigger removes all the customers.

Each queue is modelled by setting the transition corresponding to arrival events (positive customer or triggers) as passive, and those corresponding to job completion events as active. The label synchronising a departure from  $S_i$  with arrival as a customer to  $S_j$  is  $a_{i,j}$ . If a movement from  $S_i$  to  $S_j$  may occur both as a positive customer and as a trigger we use  $a_{i,j}^+$  and  $a_{i,j}^-$ , respectively.

Since all the models in the network have been studied in literature, one could apply RCAT and derive the traffic equation system. Note that this can be done thanks to the novelty of the modular approach to product-form introduced by RCAT (otherwise one should prove the product-form as solution of the GBE). Let  $x_{i,j}$  be the rate associated with the passive transitions with label  $a_{i,j}$ , and let  $\rho_n$  be the load factor of station  $n$ . Then, after setting the following values for the load factors:



$$\begin{aligned}
\rho_1 &= \frac{\lambda + x_{2,1} + x_{4,1} + \mu_1 + x_{9,1}}{2\mu_1} - \frac{1}{2\mu_1} \left( (\lambda + x_{2,1} + x_{4,1})^2 + \mu_1^2 + x_{9,1}^2 \right. \\
&\quad \left. + 2(\lambda + x_{2,1} + x_{4,1})x_{9,1} + 2\mu_1 x_{9,1} - 2(\lambda + x_{2,1} + x_{4,1})\mu_1 \right)^{\frac{1}{2}} \\
\rho_2 &= x_{1,2}/\mu_2 \\
\rho_3 &= x_{1,3}/\mu_3 \\
\rho_4 &= x_{1,4}/\mu_4 \\
\rho_5 &= (x_{2,5} + x_{3,5})/(\mu_5 + x_{1,5}^- + x_{8,5}^-) \\
\rho_6 &= x_{3,6}/(\mu_6 + x_{2,6}^-) \\
\rho_7 &= (x_{3,7} + x_{4,7})/(\mu_7 + x_{10,7}^-) \\
\rho_8 &= \frac{x_{2,8} + x_{5,8} + x_{6,8}^+ + \mu_8 + x_{6,8}^-}{2\mu_8} - \frac{1}{2\mu_8} \left( (x_{2,8} + x_{5,8} + x_{6,8}^+)^2 + \mu_8^2 + x_{6,8}^-^2 \right. \\
&\quad \left. + 2(x_{2,8} + x_{5,8} + x_{6,8}^+)x_{6,8}^- + 2\mu_8 x_{6,8}^- - 2(x_{2,8} + x_{5,8} + x_{6,8}^+)\mu_8 \right)^{\frac{1}{2}} \\
\rho_9 &= \frac{x_{8,9} + \mu_9 + x_{10,9}^- - \left( x_{8,9}^2 + \mu_9^2 + x_{10,9}^-^2 + 2x_{8,9}x_{10,9}^- + 2\mu_9 x_{10,9}^- - 2x_{8,9}\mu_9 \right)^{\frac{1}{2}}}{2\mu_9} \\
\rho_{10} &= \frac{x_{6,10}^+ + x_{7,10} + \mu_{10} + x_{6,10}^-}{2\mu_{10}} - \frac{1}{2\mu_{10}} \left( (x_{6,10}^+ + x_{7,10})^2 + \mu_{10}^2 + x_{6,10}^-^2 \right. \\
&\quad \left. + 2(x_{6,10}^+ + x_{7,10})x_{6,10}^- + 2\mu_{10}x_{6,10}^- - 2(x_{6,10}^+ + x_{7,10})\mu_{10} \right)^{\frac{1}{2}},
\end{aligned}$$

according to RCAT, the traffic equations are:

$$x_{i,j} = \rho_i \mu_i r_{i,j}^c$$

for every  $i, j$  where  $c \in \{+, -\}$  and  $r_{i,j}^c > 0$ .

We could try to solve the model, i.e., to find all unknown rates  $x_{i,j}$  by feeding the traffic equations system to a Computer Algebra System (CAS) software, e.g., Mathematica. However, in our tests this computation has shown to be unfeasible even for relatively small models since the system is not linear.

Using our algorithm, the computation of the numerical solutions for the system, described as a set of cooperating CTMCs, is straightforward. Given the model description, we set the truncation operator at  $n$  states, using the formula, derived from the condition  $\pi(n) \leq \tau$ :

$$n = \begin{cases} \text{MAX\_STATES} & \text{if } \rho \geq 1 \\ \max \left( \text{MIN\_STATES}, \min \left( \text{MAX\_STATES}, \left\lceil \frac{\ln \tau - \ln(1 - \rho)}{\ln \rho} \right\rceil \right) \right) & \text{otherwise,} \end{cases}$$

where MIN\_STATES and MAX\_STATES are user-defined constants that determine the minimum and maximum number of states allowed for model truncation, respectively.

Given a set of input parameters  $\lambda, \mu_i, \mathbf{R}^+, \mathbf{R}^-$  such that  $\forall i \in \{1 \dots 10\}, \rho_i < 1$ , i.e., the system is stable, in our tests the algorithm has always converged in less than 10 iterations for precision  $\varepsilon = 10^{-5}$  and  $\tau = 10^{-5}$ . For example, consider the following routing matrices:

$$\mathbf{R}^+ = \begin{pmatrix} 0 & 0.2 & 0.3 & 0.4 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.1 & 0 & 0 & 0 & 0.2 & 0 & 0 & 0.2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.3 & 0.5 & 0.2 & 0 & 0 & 0 \\ 0.3 & 0 & 0 & 0 & 0 & 0 & 0.7 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.3 & 0 & 0.5 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix},$$

$$\mathbf{R}^- = \begin{pmatrix} 0 & 0 & 0 & 0 & 0.1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.4 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.1 & 0 & 0.1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.1 & 0 & 0 & 0 & 0 & 0 \\ 0.1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.2 & 0 & 0.05 & 0 \end{pmatrix},$$

where the probability of a customer leaving the system after a service completion at station  $i$  is  $1 - \sum_{j=1}^{10} (r_{i,j}^+ + r_{i,j}^-)$ . In this case for values of  $\lambda$  and  $\mu_i$  shown in Table 5.1 we obtained the solutions in 6 iterations. Process sizes and  $\rho$  values are shown in Table 5.2, while computed passive rates are shown in Table 5.3.

In this example we have shown how the INAP+ algorithm can be applied to an heterogeneous system unsolvable with the INAP algorithm. We also noted that the symbolic solution of non-linear equations system that arose from the model is unfeasible. As we previously stated, though, our algorithm is not limited to queueing models, and could be used, redefining the truncation operator, whenever there is a way to determine an upper bound for stationary probabilities vector size. The algorithm has been tested for several models with different types of stations and network topologies. The results of the tests confirm what is shown in this example.

## 5.7 Conclusions

In this chapter we have presented two algorithms to detect and solve Markovian stochastic models in product-form. For models with an infinite state space we spec-

name	value
$\lambda$	5.0
$\mu_1$	4.5
$\mu_2$	4.1
$\mu_3$	4.2
$\mu_4$	4.3
$\mu_5$	4.4
$\mu_6$	4.5
$\mu_7$	4.6
$\mu_8$	4.7
$\mu_9$	4.9
$\mu_{10}$	5.0

Table 5.1: Parameters

$i$	$\rho_i$	$n$
1	0.94826	161
2	0.20816	8
3	0.30480	10
4	0.39695	12
5	0.11279	6
6	0.13221	6
7	0.29459	10
8	0.19497	7
9	0.03679	4
10	0.32479	10

Table 5.2:  $\rho$  and  $n$ 

$x_{i,j}$	value	$x_{i,j}$	value
$x_{1,2}$	0.8534	$x_{4,7}$	1.1948
$x_{1,3}$	1.2802	$x_{5,8}$	0.4963
$x_{1,4}$	1.7069	$x_{6,8}^+$	0.1785
$x_{1,5}^-$	0.4267	$x_{6,8}^-$	0.0595
$x_{2,1}$	0.0853	$x_{6,10}^+$	0.2975
$x_{2,5}$	0.1707	$x_{6,10}^-$	0.0595
$x_{2,6}^-$	0.3414	$x_{7,10}$	1.3551
$x_{2,8}$	0.2560	$x_{8,5}^-$	0.0916
$x_{3,5}$	0.3840	$x_{8,9}$	0.1833
$x_{3,6}$	0.6401	$x_{9,1}^-$	0.0180
$x_{3,7}$	0.2560	$x_{10,7}^-$	0.3248
$x_{4,1}$	0.5121	$x_{10,9}^-$	0.0812

Table 5.3: Results

ify how a dynamic truncation given an arbitrary precision could be performed. Although those algorithms share the lack of a proof of convergence with many other iterative procedures defined for performance evaluation purposes, the extensive numerical tests that have been run have not shown any false-negative result. The number of iterations to converge with a precision of  $10^{-6}$  has never exceeded 20 in all our tests.

We have also presented a tool that we have developed, based on the algorithms described above, that has proved to be able to identify and compute several product-form results based on pairwise synchronisations, such as Jackson networks, G-networks with or without catastrophes, Akyildiz's results about product-form networks with blocking and other formalisms, and a graphical interface that allows the user to define models and cooperations between them in an easy and convenient way, ignoring all the technical aspects of the machine representation of the problem.

Our current effort is in developing an extension of the tool in order to reach three objectives: 1) allow for the specification of models with multiple incoming active transitions, exploiting the result presented in [133], 2) allow for the specification of models with multiple outgoing passive transitions with the same label, and 3) allow for the graphical specification of models with regular but infinite structure, as those tractable by the algorithm described in paragraph 5.3. For all these three objectives we already have a testing implementation. The current challenge consists in designing a graphical formalism capable of expressing truncation operators in an user-friendly way.

---

# 6

## Component-wise state space reduction

### 6.1 Introduction

As stated before, although compositionality is a key-feature of most of the performance formalisms and allows the modeller to combine several (possibly simple) components to form a complex architecture, the derivation of performance indices may be very time and space consuming since the state space cardinality of these kind of models tends to grow exponentially with the number of system components.

While in the previous chapter we have seen how product-form decomposition could help to mitigate this problem, here we first focus on aggregation techniques, in which the state space of components, described in a similar fashion to the one used in the previous chapter, is reduced according to some equivalence rules. The differences among our approach, classical *lumping* and *strong equivalence* will be explained thorough the chapter.

Moreover, in Chapter 7, we will show how aggregation and product-form decomposition techniques are indeed related, as we will introduce a novel concept of *conditional product form*, based on the aggregation, according to the rules described in this chapter, of reversed processes.

#### 6.1.1 Related work

The application of aggregation and lumpability techniques has been proposed to cope with the solution of models with a large state space, and it has been widely applied for various formalisms, e.g. exact and approximate aggregation in queueing networks [18, 46], decomposability and lumpability for Markov chains [112, 164], aggregation of stochastic Petri nets [44], stochastic automata or Markovian process algebras [101, 83], where the references should be considered just as examples of remarkable work in the corresponding field.

As concerns lumpability, under certain conditions the state space of a Markov chain can be partitioned into subsets of states, each of which can be seen as a single state of a smaller Markov chain. The original chain is then said to be *lumpable*. The

process of lumping states in a Markov chain [112] defines a state space partition of the Markov chain and a corresponding new lumped process with a reduced state space. Specifically, consider a continuous-time homogeneous Markov chain (CTMC) having a state space  $S$  with  $n$  states and transition rate matrix  $\mathbf{Q}$ . Let  $\tilde{s}_1, \dots, \tilde{s}_N$  be a partition of space  $S$ , where usually  $N \ll n$ . The CTMC is lumpable with respect to the partition if for any subset  $\tilde{s}_i$  and states  $s, s' \in \tilde{s}_i$ ,  $\sum_{s'' \in \tilde{s}_k} \mathbf{Q}(s, s'') = \sum_{s'' \in \tilde{s}_k} \mathbf{Q}(s', s'')$  for  $0 \leq k \leq N$ . That is, for any two states in a given subset the cumulative transition rate to any other partition is equal. Efficient and optimal algorithms for lumping CTMCs are presented in [2]; here, the symmetries of the CTMCs underlying complex models are exploited to carry out a reduction of the state space using different notions of lumping. Conversely, in the approach we propose in this paper, the lumping is done component-wise and not on the joint CTMC. It is easy to see that a component-wise lumping implies a lumping of the underlying CTMC but the opposite is not true. Nevertheless, working at component-level allows us to deal with smaller state spaces and hence improve the computational time and space costs.

As concerns aggregation to analyse complex systems, several approaches consider hierarchical decomposition of the model into a set of submodels. Such a decomposition-aggregation approach defines three steps: 1) partition of the original model into a set of sub-models, and analysis of each sub-model in isolation; 2) definition of a new and smaller aggregated model where each component represents an aggregated sub-model; 3) analysis of the aggregated model. Exact aggregation defines the new aggregated model, that can be proved to be equivalent to the original one, i.e., with the same solution for a set of performance indices, usually the aggregated stationary state distribution. Unfortunately, exact aggregation algorithms on the Markov chain have a computational complexity that is comparable to that of the solution of the entire model [62]. However, some exact aggregation methods have been defined directly in terms of model components at a higher level of abstraction. Moreover, under special constraints, conditions for exact aggregation have been defined for various classes of Markov models and for product-form models, such as product-form queueing networks [49, 18]. Several approximate methods based on decomposition and aggregation, such as those described in [62, 163, 61, 75], have also been proposed in the literature. Note that step (1) that defines the model state partition is non-trivial and affects the quality of the analysis results. However, some criteria and observations depending on the type of system, can help this choice. Some interesting methods applied to nearly-completely decomposable models were proposed by Courtois [62] to derive bounded approximation of the steady-state solution of the Markov chain, whose condition is given in terms of model (queueing network) parameters. Stewart proposed a different bounded aggregation methods for Markov chains [163], whose error analysis is based on vector and matrix norms. The bound method by Courtois and Semal [61] has been further applied for solving Markov chains and to derive bounds on the steady-state solution of quasi-lumpable Markov chains [75].

### 6.1.2 Contribution

In this chapter, we are interested in studying those formalisms that allow the modeller to describe a system in a modular way. These formalisms can be widely applied in practice because they conform to good engineering principles. For instance PEPA or stochastic automata networks [146] represent important examples of formalisms that yield a high modularity. In [101, 83] the authors present an exact technique to improve performance evaluation in PEPA models based on a relation called *strong equivalence*. The idea is to exploit the intrinsic modular nature of PEPA models. In [101] the author discusses the relations of the strong equivalence with the previous results on Markov chain lumping. Following this idea, in considering a model defined in terms of a set of cooperating components, we aim at applying the notion of lumping at the component level rather than at the CTMC level of the joint process. As in [101, 83], we focus on stochastic performance models with underlying CTMCs. With respect to the cited papers we give a notion of lumpability which is more general and we present and prove two theorems on lumping in cooperating stochastic models both for the original model and for the time-reversed automata. Reversed processes have been known to be related to model decompositions especially in case of product-form models (see e.g., [111, 89]). Here, we show that time-reversed automata can be used also in lumpability. In particular, we observe that a class of product-form models can be seen as a special case of the results we present here. This chapter extends the results presented in [12] as follows. First, we relax the conditions of the theorem on the forward automaton lumping and give a proof that does not require the underlying assumption of product-form as that given in [12]. Moreover, the condition of Theorem 3 are more general than those required by the analogous presented in [12]. Second, we further investigate the properties of the lumping of reversed automata in case of non-product-forms. In this context, we prove that under a reversibility assumption on the lumping of the reversed process, a *conditional* product-form solution can be defined. As far as we know, a similar approach is that proposed in [46], called *higher order product-forms* as a mean of approximating the steady-state probabilities of a set of cooperating components. However, here we deal with exact solutions and the conditions are interpreted in terms of lumping of the cooperating automata. A more similar theory is proposed by Economou in [69] related to Markov modulated processes. The conditional product-form conditions proposed by Economou may be interpreted in terms of lumping of cooperating automata as shown by Theorem 4.

In the following parts we will often refer to formalisms described in section 2.4.

## 6.2 Exact lumpability

In this section we prove Theorem 3 that can be applied to define efficient algorithms for computing the marginal steady-state probability distribution for one of

the cooperating models based on the exact lumping of the forward or reversed processes underlying the other component. Observe that, in this context, the concept of lumpability as introduced in [112] is extended in order to take into account the synchronising transition types. Roughly speaking, we aim at replacing component  $M_1$  by a smaller one denoted by  $\tilde{M}_1$  such that the marginal steady-state probability distribution of  $M_2$  in the cooperation  $\tilde{M}_1 \otimes M_2$  is identical to that of  $M_2$  in the cooperation  $M_1 \otimes M_2$ . In queueing theory, this idea has previously been applied for defining algorithms for approximate analysis of queueing networks (see, among others, [101, 94, 45] and the reference therein). However, Theorem 3 and 4 give sufficient conditions for deriving exact automaton lumping similarly to what is done in [101, 83].

The following definition plays a pivotal role in what follows and extends the concept of lumpability in order to deal with synchronising transition types. Note that, in what follows, without loss of generality, we assume that  $M_1$  is the automaton to be lumped in the cooperation  $M_1 \otimes M_2$ .

**Definition 3** (Exact lumped automata). *Given automaton  $M_1$ , a set of transition types  $\mathcal{T}$ , and a partition of the states of  $M_1$  into  $\tilde{N}_1$  clusters  $\mathcal{S} = \{\tilde{1}, \tilde{2}, \dots, \tilde{N}_1\}$ , we say that  $\mathcal{S}$  is an exact lumping for  $M_1$  if it is possible to define a set of functions  $\tilde{\varphi}_1^\ell : \mathcal{S} \times \mathcal{S} \rightarrow \mathbb{R}^+$  such that:*

1.  $\forall \tilde{s}_1, \tilde{s}'_1 \in \mathcal{S}, \tilde{s}'_1 \neq \tilde{s}_1, \forall s_1 \in \tilde{s}_1 \quad \sum_{s'_1 \in \tilde{s}'_1} \mathbf{E}_{11}(s_1, s'_1) = \tilde{\varphi}_1^1(\tilde{s}_1, \tilde{s}'_1)$
2.  $\forall \ell > 2, \forall \tilde{s}_1, \tilde{s}'_1 \in \mathcal{S}, \forall s_1 \in \tilde{s}_1 \quad \sum_{s'_1 \in \tilde{s}'_1} \mathbf{E}_{1\ell}(s_1, s'_1) = \tilde{\varphi}_1^\ell(\tilde{s}_1, \tilde{s}'_1).$

If  $M_1$  is lumpable with respect to  $\mathcal{S}$ , we define the automaton  $\tilde{M}_1$  with  $\tilde{N}_1$  states as follows:

$$\begin{aligned} \tilde{\mathbf{E}}_{11}(\tilde{s}_1, \tilde{s}'_1) &= \begin{cases} \tilde{\varphi}_1^1(\tilde{s}_1, \tilde{s}'_1) & \text{if } \tilde{s}_1 \neq \tilde{s}'_1 \\ 0 & \text{otherwise} \end{cases} \\ \tilde{\mathbf{E}}_{12} &= \mathbf{I}, \quad \tilde{\mathbf{E}}_{1\ell}(\tilde{s}_1, \tilde{s}'_1) = \tilde{\varphi}_1^\ell(\tilde{s}_1, \tilde{s}'_1) \quad t > 2 \end{aligned}$$

where  $\tilde{\lambda}_\ell = \lambda_\ell$  for all  $\ell$  are the rates associated with the transition types in the cooperation between  $\tilde{M}_1$  and  $M_2$ .

As one may expect, if  $\tilde{M}_1$  is an exact lumped automaton of  $M_1$ , then the CTMC underlying  $\tilde{M}_1$  is an exact lumping of that of  $M_1$  in the standard sense of [112] but the opposite is not true (e.g., consider the different role of synchronising and non-synchronising transitions).

In what follows we assume that  $M_1$ ,  $M_2$  and their cooperation to have ergodic underlying CTMCs (in the case of the cooperation, the state space is the Cartesian product of the state spaces of the single components). We refer to this as the *ergodicity assumption*.



**Theorem 3.** *Given the model defined as  $M_1 \otimes M_2$ , let  $\tilde{M}_1$  be an exact lumping of  $M_1$  whose clusters are  $\mathcal{S} = \{\tilde{1}, \dots, \tilde{N}_1\}$ . Then, under ergodicity assumptions, the following relation between the steady-state probabilities  $\pi(s_1, s_2)$  of  $M_1 \otimes M_2$  and  $\tilde{\pi}(\tilde{s}_1, s_2)$  of  $\tilde{M}_1 \otimes M_2$  holds:*

$$\forall \tilde{s}_1 = \tilde{1}, \dots, \tilde{N}_1, \forall s_2 = 1, \dots, N_2, \tilde{\pi}(\tilde{s}_1, s_2) = \sum_{s_1 \in \tilde{s}_1} \pi(s_1, s_2). \quad (6.1)$$

Note that Equation (6.1) of Theorem 3 follows from the notion of strong-equivalence of PEPA agents [101]. However, the following proof is needed since Definition 3 is more general than that of strong equivalence as discussed in Section 6.2.1.

*Proof of Theorem 3.* The proof is based on the following idea: for a general cluster  $\tilde{s}_1$  of  $\tilde{M}_1$  we sum the GBE of states  $(s_1, s_2)$  of  $M_1 \otimes M_2$  thus obtaining an identity. Then, by substitution of Equation (6.1) in the system of GBE of  $\tilde{M}_1 \otimes M_2$  deriving this identity and by the uniqueness of the steady-state distribution we prove the theorem.

The global balance equation for a generic state  $(s_1, s_2)$  of  $M_1 \otimes M_2$  is:

$$\begin{aligned} \pi(s_1, s_2) & \left( \sum_{s'_1=1}^{N_1} \lambda_1 \mathbf{E}_{11}(s_1, s'_1) + \sum_{s'_2=1}^{N_2} \lambda_2 \mathbf{E}_{22}(s_2, s'_2) \right. \\ & \quad \left. + \sum_{\ell=3}^T \sum_{s'_1=1}^{N_1} \sum_{s'_2=1}^{N_2} \lambda_\ell \mathbf{E}_{1\ell}(s_1, s'_1) \mathbf{E}_{2\ell}(s_2, s'_2) \right) \\ & = \sum_{s'_1=1}^{N_1} \pi(s'_1, s_2) \lambda_1 \mathbf{E}_{11}(s'_1, s_1) + \sum_{s'_2=1}^{N_2} \pi(s_1, s'_2) \lambda_2 \mathbf{E}_{22}(s'_2, s_2) \\ & \quad + \sum_{\ell=3}^T \sum_{s'_1=1}^{N_1} \sum_{s'_2=1}^{N_2} \pi(s'_1, s'_2) \lambda_\ell \mathbf{E}_{1\ell}(s'_1, s_1) \mathbf{E}_{2\ell}(s'_2, s_2). \quad (6.2) \end{aligned}$$

Analogously, the global balance equation for a generic state  $(\tilde{s}_1, s_2)$  of  $\tilde{M}_1 \otimes M_2$  is:

$$\begin{aligned}
\tilde{\pi}(\tilde{s}_1, s_2) & \left( \sum_{\tilde{s}'_1=\bar{1}}^{\tilde{N}_1} \tilde{\lambda}_1 \tilde{\mathbf{E}}_{11}(\tilde{s}_1, \tilde{s}'_1) + \sum_{s'_2=1}^{N_2} \lambda_2 \mathbf{E}_{22}(s_2, s'_2) \right. \\
& \quad \left. + \sum_{\ell=3}^T \sum_{\tilde{s}'_1=\bar{1}}^{\tilde{N}_1} \sum_{s'_2=1}^{N_2} \tilde{\lambda}_\ell \tilde{\mathbf{E}}_{1\ell}(\tilde{s}_1, \tilde{s}'_1) \mathbf{E}_{2\ell}(s_2, s'_2) \right) \\
& = \sum_{\tilde{s}'_1=\bar{1}}^{\tilde{N}_1} \tilde{\pi}(\tilde{s}'_1, s_2) \tilde{\lambda}_1 \tilde{\mathbf{E}}_{11}(\tilde{s}'_1, \tilde{s}_1) + \sum_{s'_2=1}^{N_2} \tilde{\pi}(\tilde{s}_1, s'_2) \lambda_2 \mathbf{E}_{22}(s'_2, s_2) \\
& \quad + \sum_{\ell=3}^T \sum_{\tilde{s}'_1=\bar{1}}^{\tilde{N}_1} \sum_{s'_2=1}^{N_2} \pi(\tilde{s}'_1, s'_2) \tilde{\lambda}_\ell \tilde{\mathbf{E}}_{1\ell}(\tilde{s}'_1, \tilde{s}_1) \mathbf{E}_{2\ell}(s'_2, s_2). \quad (6.3)
\end{aligned}$$

Summing both the members of Equation (6.2) over states  $s_1$  belonging to  $\tilde{s}_1$  of  $\tilde{M}_1$ , we obtain the following identity:

$$\begin{aligned}
\sum_{s_1 \in \tilde{s}_1} & \left[ \pi(s_1, s_2) \left( \sum_{s'_1=1}^{N_1} \lambda_1 \mathbf{E}_{11}(s_1, s'_1) + \sum_{s'_2=1}^{N_2} \lambda_2 \mathbf{E}_{22}(s_2, s'_2) \right. \right. \\
& \quad \left. \left. + \sum_{\ell=3}^T \sum_{s'_1=1}^{N_1} \sum_{s'_2=1}^{N_2} \lambda_\ell \mathbf{E}_{1\ell}(s_1, s'_1) \mathbf{E}_{2\ell}(s_2, s'_2) \right) \right] \\
& = \sum_{s_1 \in \tilde{s}_1} \sum_{s'_1=1}^{N_1} \pi(s'_1, s_2) \lambda_1 \mathbf{E}_{11}(s'_1, s_1) + \sum_{s_1 \in \tilde{s}_1} \sum_{s'_2=1}^{N_2} \pi(s_1, s'_2) \lambda_2 \mathbf{E}_{22}(s'_2, s_2) \\
& \quad + \sum_{s_1 \in \tilde{s}_1} \sum_{\ell=3}^T \sum_{s'_1=1}^{N_1} \sum_{s'_2=1}^{N_2} \pi(s'_1, s'_2) \lambda_\ell \mathbf{E}_{1\ell}(s'_1, s_1) \mathbf{E}_{2\ell}(s'_2, s_2). \quad (6.4)
\end{aligned}$$

We consider the left-hand-side of Equation (6.4), and use the following relations that follow from the hypothesis of lumpability of  $M_1$  into  $\tilde{M}_1$  and Definition 3:

$$\begin{aligned}
& \sum_{s_1 \in \tilde{s}_1} \pi(s_1, s_2) \sum_{s'_1=1}^{N_1} \lambda_1 \mathbf{E}_{11}(s_1, s'_1) = \sum_{s_1 \in \tilde{s}_1} \pi(s_1, s_2) \sum_{\tilde{s}'_1=\bar{1}}^{\tilde{N}_1} \sum_{s'_1 \in \tilde{s}'_1} \lambda_1 \mathbf{E}_{11}(s_1, s'_1) \\
& = \left( \sum_{s_1 \in \tilde{s}_1} \pi(s_1, s_2) \right) \sum_{\substack{\tilde{s}'_1=\bar{1} \\ \tilde{s}'_1 \neq \tilde{s}_1}}^{\tilde{N}_1} \tilde{\lambda}_1 \tilde{\mathbf{E}}_{11}(\tilde{s}_1, \tilde{s}'_1) + \sum_{s_1 \in \tilde{s}_1} \sum_{s'_1 \in \tilde{s}_1} \pi(s_1, s_2) \lambda_1 \mathbf{E}_{11}(s_1, s'_1),
\end{aligned}$$

and:

$$\begin{aligned}
& \sum_{s_1 \in \tilde{s}_1} \pi(s_1, s_2) \sum_{\ell=3}^T \sum_{s'_1=1}^{N_1} \sum_{s'_2=1}^{N_2} \lambda_\ell \mathbf{E}_{1\ell}(s_1, s'_1) \mathbf{E}_{2\ell}(s_2, s'_2) \\
&= \sum_{s_1 \in \tilde{s}_1} \pi(s_1, s_2) \sum_{\ell=3}^T \sum_{s'_2=1}^{N_2} \mathbf{E}_{2\ell}(s_2, s'_2) \sum_{\tilde{s}'_1=\bar{1}}^{\tilde{N}_1} \sum_{s'_1 \in \tilde{s}'_1} \lambda_\ell \mathbf{E}_{1\ell}(s_1, s'_1) \\
&= \left( \sum_{s_1 \in \tilde{s}_1} \pi(s_1, s_2) \right) \sum_{\ell=3}^T \sum_{s'_2=1}^{N_2} \mathbf{E}_{2\ell}(s_2, s'_2) \sum_{\tilde{s}'_1=\bar{1}}^{\tilde{N}_1} \tilde{\lambda}_\ell \tilde{\mathbf{E}}_{1\ell}(\tilde{s}_1, \tilde{s}'_1),
\end{aligned}$$

We analogously transform the terms in right-hand-side of Equation (6.4):

$$\begin{aligned}
& \sum_{s_1 \in \tilde{s}_1} \sum_{s'_1=1}^{N_1} \pi(s'_1, s_2) \lambda_1 \mathbf{E}_{11}(s'_1, s_1) = \sum_{s_1 \in \tilde{s}_1} \sum_{\tilde{s}'_1=\bar{1}}^{\tilde{N}_1} \sum_{s'_1 \in \tilde{s}'_1} \pi(s'_1, s_2) \lambda_1 \mathbf{E}_{11}(s'_1, s_1) \\
&= \sum_{\tilde{s}'_1=\bar{1}}^{\tilde{N}_1} \sum_{s'_1 \in \tilde{s}'_1} \pi(s'_1, s_2) \sum_{s_1 \in \tilde{s}_1} \lambda_1 \mathbf{E}_{11}(s'_1, s_1) \\
&= \sum_{\substack{\tilde{s}'_1=\bar{1} \\ \tilde{s}'_1 \neq \tilde{s}_1}}^{\tilde{N}_1} \left( \sum_{s'_1 \in \tilde{s}'_1} \pi(s'_1, s_2) \right) \tilde{\lambda}_1 \tilde{\mathbf{E}}_{11}(\tilde{s}'_1, \tilde{s}_1) + \sum_{s'_1 \in \tilde{s}_1} \sum_{s_1 \in \tilde{s}_1} \pi(s'_1, s_2) \lambda_1 \mathbf{E}_{11}(s'_1, s_1),
\end{aligned}$$

and:

$$\begin{aligned}
& \sum_{s_1 \in \tilde{s}_1} \sum_{\ell=3}^T \sum_{s'_1=1}^{N_1} \sum_{s'_2=1}^{N_2} \pi(s'_1, s'_2) \lambda_\ell \mathbf{E}_{1\ell}(s'_1, s_1) \mathbf{E}_{2\ell}(s'_2, s_2) \\
&= \sum_{\ell=3}^T \sum_{\tilde{s}'_1=\bar{1}}^{\tilde{N}_1} \sum_{s'_1 \in \tilde{s}'_1} \sum_{s'_2=1}^{N_2} \pi(s'_1, s'_2) \mathbf{E}_{2\ell}(s'_2, s_2) \sum_{s_1 \in \tilde{s}_1} \lambda_\ell \mathbf{E}_{1\ell}(s'_1, s_1) \\
&= \sum_{\ell=3}^T \sum_{\tilde{s}'_1=\bar{1}}^{\tilde{N}_1} \sum_{s'_2=1}^{N_2} \left( \sum_{s_1 \in \tilde{s}_1} \pi(\tilde{s}'_1, s'_2) \right) \mathbf{E}_{2\ell}(s'_2, s_2) \tilde{\lambda}_\ell \tilde{\mathbf{E}}_{1\ell}(\tilde{s}'_1, \tilde{s}_1).
\end{aligned}$$

Noting that trivially:

$$\sum_{s_1 \in \tilde{s}_1} \sum_{s'_1 \in \tilde{s}_1} \pi(s_1, s_2) \lambda_1 \mathbf{E}_{11}(s_1, s'_1) = \sum_{s'_1 \in \tilde{s}_1} \sum_{s_1 \in \tilde{s}_1} \pi(s'_1, s_2) \lambda_1 \mathbf{E}_{11}(s'_1, s_1),$$

and that  $\tilde{\mathbf{E}}_{11}(\tilde{s}_1, \tilde{s}_1) = 0$  by Definition 3, Equation (6.4) can be rewritten as follows:

$$\begin{aligned}
& \left( \sum_{s_1 \in \tilde{s}_1} \pi(s_1, s_2) \right) \left( \sum_{\tilde{s}'_1 = \bar{1}}^{\tilde{N}_1} \tilde{\lambda}_1 \tilde{\mathbf{E}}_{11}(\tilde{s}_1, \tilde{s}'_1) + \sum_{s'_2=1}^{N_2} \lambda_2 E_{22}(s_2, s'_2) \right. \\
& \quad \left. + \sum_{\ell=3}^T \sum_{\tilde{s}'_1 = \bar{1}}^{\tilde{N}_1} \sum_{s'_2=1}^{N_2} \tilde{\lambda}_\ell \tilde{\mathbf{E}}_{1\ell}(\tilde{s}_1, \tilde{s}'_1) \mathbf{E}_{2\ell}(s_2, s'_2) \right) \\
& = \sum_{\tilde{s}'_1 = \bar{1}}^{\tilde{N}_1} \left( \sum_{s'_1 \in \tilde{s}'_1} \pi(s'_1, s_2) \right) \tilde{\lambda}_1 \tilde{\mathbf{E}}_{11}(\tilde{s}'_1, \tilde{s}_1) + \sum_{s'_2=1}^{N_2} \left( \sum_{s_1 \in \tilde{s}_1} \pi(s_1, s'_2) \right) \lambda_2 \mathbf{E}_{22}(s'_2, s_2) \\
& \quad + \sum_{\ell=3}^T \sum_{\tilde{s}'_1 = \bar{1}}^{\tilde{N}_1} \sum_{s'_2=1}^{N_2} \left( \sum_{s_1 \in \tilde{s}_1} \pi(\tilde{s}'_1, s'_2) \right) \tilde{\lambda}_\ell \tilde{\mathbf{E}}_{1\ell}(\tilde{s}'_1, \tilde{s}_1) \mathbf{E}_{2\ell}(s'_2, s_2) \quad (6.5)
\end{aligned}$$

The theorem is proved by observing that the substitution of  $\tilde{\pi}(\tilde{s}_1, s_2)$  in Equation (6.3) by the expression given in Theorem 3 straightforwardly produces Equation (6.5) which is an identity.  $\square$

Corollary 1 states an efficient way of computing the marginal distribution of  $M_2$ .

**Corollary 1.** *Let  $M_1$  and  $M_2$  be two automata, and let  $\tilde{M}_1$  be an exact automaton lumping of  $M_1$ . Then, under ergodicity assumption, the marginal steady-state probability distribution  $\pi_2(s_2)$  of  $M_2$  is:*

$$\pi_2(s_2) = \sum_{\tilde{s}_1 = \bar{1}}^{\tilde{N}_1} \tilde{\pi}(\tilde{s}_1, s_2),$$

where  $\tilde{\pi}(\tilde{s}_1, s_2)$  is the stationary distribution of  $\tilde{M}_1 \otimes M_2$ .

The proof immediately follows from Theorem 3.

**Example 8** (Exact automaton lumping). *We consider the automaton depicted by Figure 6.1. According to Definition 3 we derive the lumped automaton of Figure 6.2.*

### 6.2.1 Exact lumping and strong equivalence

The reader familiar with process algebra can observe that Definition 3 is closely related to the definition of *strong equivalence* between PEPA processes given in [101]. The author exploits this approach with the same aims that we have here.

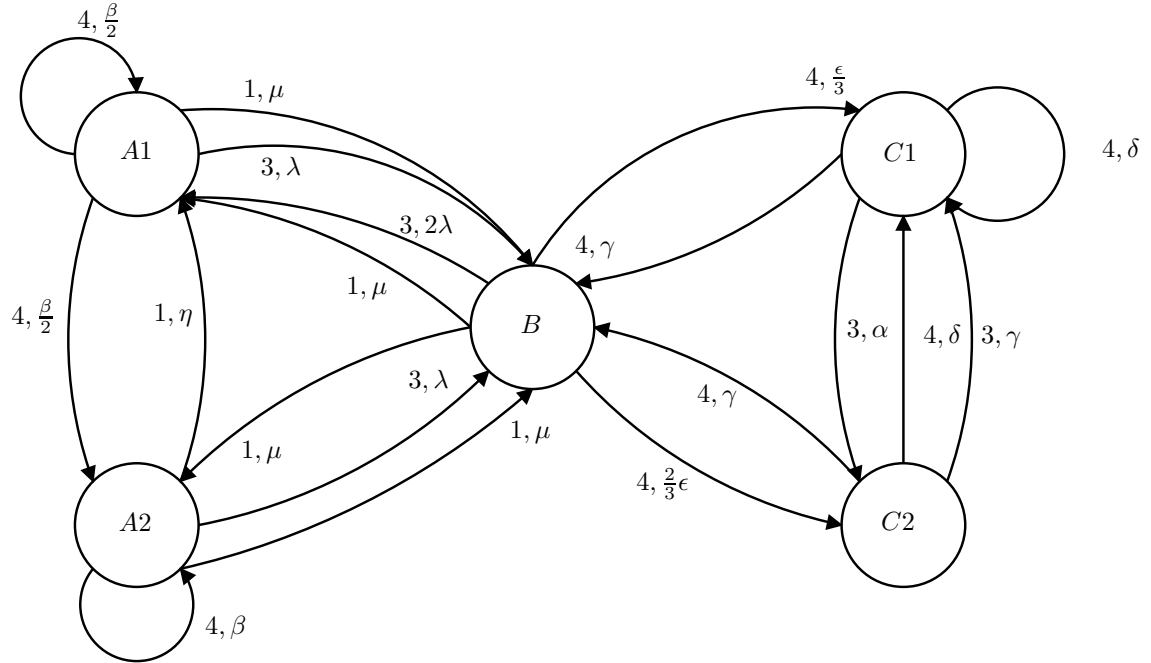


Figure 6.1: Example of automaton with types  $T = \{1, 2, 3, 4\}$ . Arcs are labelled by their type  $\ell$  and the rate  $\lambda_\ell \mathbf{E}_{1\ell}(s_1, s'_1)$ . Self-loops associated with type 2 are omitted for the sake of clarity.

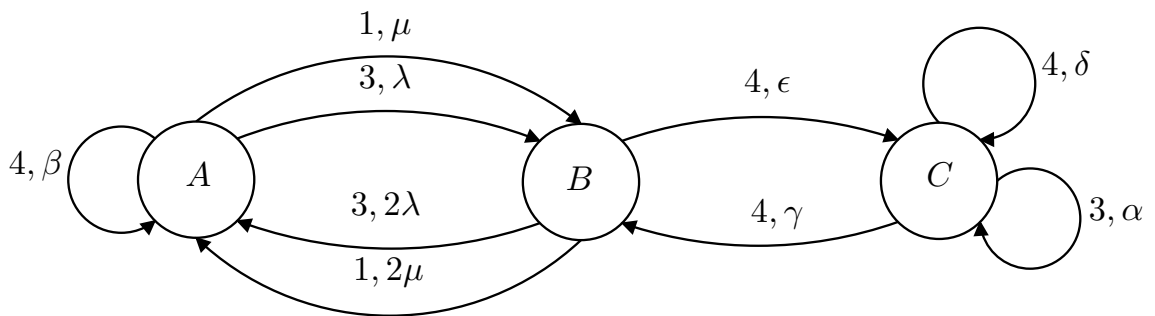


Figure 6.2: Lumping of the automaton of Figure 6.1. Arcs are labelled by their type  $\ell$  and the rate  $\hat{\lambda}_\ell \hat{\mathbf{E}}_{1\ell}$ . Self-loops associated with type 2 are omitted for the sake of clarity.

The difference between Definition 3 and the concept of *strong equivalence* concerns the conditions about the non-synchronising transitions, i.e., those that in PEPA are called  $\tau$ -actions (in our framework these correspond to transitions with type  $t = 1$ ). In fact, Definition 3 distinguishes between non-synchronising ( $t = 1$ ) and synchronising ( $t > 2$ ) transition types, as the former does not need to have constant outgoing rate from a state of a cluster to other states of the same cluster. Therefore, we can say that strong equivalence implies lumping in the sense of Definition 3 but not vice versa, as illustrated in Example 8, where states  $A_1$  and  $A_2$  are not strong equivalent because of the transition of type 1 and rate  $\eta$  from state  $A_2$  to state  $A_1$ . It is worth noting that while our approach deals only with cooperations between automata, PEPA's strong equivalence is a congruence which is preserved by all the process algebra's combinators.

### 6.3 Conclusions

In this chapter we have shown a set of conditions for state space aggregation which are weaker than those required by PEPA strong equivalence [101], but still preserve their compositionality with respect to synchronisations. In the next chapter we will further delve into aggregation issues, and we will show some relations between lumping and product-form theory.

# Conditional Product-Forms

## 7.1 Introduction

In Chapter 5 and 6 we have seen two of the main approaches for reducing the complexity of the solution of cooperating stochastic models, i.e., product forms and state space aggregations. This chapter proves some interesting relations between the joint process  $M_1 \otimes M_2$  and the lumping of the automaton which is the time-reversed of  $M_1$ . Using time-reversibility to study the steady-state distribution of a complex model is not a new result itself (see, e.g., the results on stochastic networks and product-forms presented in [111, 89]); nevertheless, the results proved here are novel: indeed, we show how *conditional product-form* can be interpreted as a lumping of reversed automata, thus revealing an unexpected link between the results of the two previous chapters. Note that in [46] higher order product-forms are used only as a mean of approximating the stationary distribution of the joint process, without showing when they provide exact results.

The following section introduces some definitions that will be useful in the remaining of the chapter.

### 7.1.1 Feed-forward synchronisations

The main restriction we consider in this work concerns the class of synchronisations that we admit in our model. Roughly speaking we require the automaton that has to be lumped to have a marginal steady-state distribution that is independent of the state of  $M_2$  (but obviously not vice-versa).

**Definition 4** (Non-blocking synchronisation). *We say that type  $\ell \in \mathcal{T}$  is non-blocking if for at least one of the cooperating automata  $M_1$  and  $M_2$  it holds that  $R_{k\ell}(s) = 1$  for all  $s = 1, \dots, N_k$ . In this case we say that  $\ell$  is active in  $M_h$ , with  $h \neq k$ , and passive in  $M_k$ ,  $k, h \in \{1, 2\}$ .*

Informally, we can say that in a non-blocking synchronisation, one of the two cooperating automata (the active with respect to  $\ell$ ) can carry out its activity of type  $\ell$  independently of the current state of the other automaton. In this case we

can say that the transition rate of active automaton  $M_h$  from state  $s_h$  to  $s'_h$  is given by:

$$q_\ell(s_h, s'_h) = \lambda_\ell \mathbf{E}_{h\ell}(s_h, s'_h).$$

Note that, by definition,  $q_1(s_1, s'_1)$  ( $q_2(s_2, s'_2)$ ) denotes the rates of the independent transitions in  $M_1$  ( $M_2$ ). As an instance, if we consider a tandem of two queues, and let  $\ell$  be the synchronisation between the customer departures from the first queue and the arrivals at the second, then a sufficient condition for the synchronisation to be non-blocking is that the second queue has infinite buffer size (see Example 1).

The following definition is needed to avoid cycles among model synchronisation. In queueing theory this corresponds to the possibility of defining queueing networks with a feed-forward structure.

**Definition 5** (Feed-forward synchronisation). *We say that the synchronisation between  $M_1$  and  $M_2$  is feed-forward if for all  $\ell \in \mathcal{T}$ ,  $\ell \neq 2$ , matrices  $E_{2\ell}$  are stochastic. We call  $M_1$  and  $M_2$  the active and passive model, respectively.*

**Remark 1.** *Observe that in a feed-forward synchronisation with non-blocking synchronisation, the infinitesimal generator underlying  $M_1$  is well-defined by:*

$$\mathbf{Q}_1 = \sum_{t=1}^T \lambda_t \mathbf{E}_{1t} - \sum_{t=1}^T \lambda_t \mathbf{D}_{1t}, \quad (7.1)$$

Hence, if  $\mathbf{Q}_1$  is associated with an ergodic CTMC, we can compute the marginal distribution of  $M_1$  in the cooperation.

Theorem 4 relies on the theory of reversed Markov processes as studied in [111] and successively in [89]. Before stating the second theorem, we briefly review some results about reversed Markov processes [111]. Given a continuous time stochastic process  $X(t)$  we say that it is *stationary* if  $(X(t_1), X(t_2), \dots, X(t_n))$  has the same joint-distribution of  $(X(t_1+\tau), X(t_2+\tau), \dots, X(t_n+\tau))$  and we say that it is *reversible* if  $(X(t_1), X(t_2), \dots, X(t_n))$  has the same joint-distribution of  $(X(\tau - t_1), X(\tau - t_2), \dots, X(\tau - t_n))$  for all  $t_1, \dots, t_n, \tau \in \mathbb{R}$  and  $n \in \mathbb{N}$ . It is easy to prove that a reversible process is also stationary. For a reversible CTMC, the following relation holds:

$$\pi_1(s_1)q_1(s_1, s'_1) = \pi_1(s'_1)q_1(s'_1, s_1),$$

where  $\pi_1(s_1)$  is the stationary probability of  $s_1$  and  $s_1, s'_1$  two arbitrary states of the CTMC, and  $q_1(s_1, s'_1)$  denotes the transition rate from  $s_1$  to  $s'_1$ . Obviously, a stationary process may be not reversible. In this case, it is still possible to define the reversed process, but the joint-distribution of  $(X(t_1), X(t_2), \dots, X(t_n))$  is in general different form that of process  $(X(\tau - t_1), X(\tau - t_2), \dots, X(\tau - t_n))$ . Assume that the forward chain admits the steady-state distribution  $\pi_1$  and has a transition from state  $s_1$  to  $s'_1$  with rate  $q(s_1, s'_1)$ , then it can be proved that the reversed process is



still a Markov process with the same state space that has a transition from  $s'_1$  to  $s_1$  whose rate  $q_1^R(s'_1, s_1)$  is given by:

$$q_1^R(s'_1, s_1) = \frac{\pi(s_1)}{\pi(s'_1)} q_1(s_1, s'_1). \quad (7.2)$$

The forward and the reversed CTMCs share the same steady-state probabilities [111]. Observe that from the transition rates of the reversed CTMC we can efficiently compute the unnormalised steady-state distribution and vice versa. Based on Equation (7.2) we give the following definition:

**Definition 6** (Timed-reversed automata). *Given the active automaton  $M_1$  synchronising on transition type  $\mathcal{T}$  with rates  $\lambda_1, \dots, \lambda_T$ , we define the timed-reversed automaton  $M_1^R$  as follows:*

$$\begin{aligned} \mathbf{E}_{1\ell}^R(s_1, s'_1) &= \frac{\pi_1(s'_1)}{\pi_1(s_1)} q_\ell(s'_1, s_1) \frac{1}{\lambda_\ell^R} \quad \ell \neq 2 \\ \mathbf{E}_{12}^R &= \mathbf{I} \end{aligned}$$

where:

$$\lambda_\ell^R = \max_{s_1=1, \dots, N_1} \left( \sum_{s'_1=1}^{N_1} q_\ell^R(s_1, s'_1) \right)$$

and

$$q_\ell^R(s_1, s'_1) = \frac{\pi_1(s'_1)}{\pi_1(s_1)} q_\ell(s'_1, s_1),$$

for all  $1 \leq s_1, s'_1 \leq N_1$  and  $\ell \neq 2$ .

**Definition 7** (Time-reversible automata). *An active automaton  $M_1$  is time-reversible if the following condition holds:*

$$\forall \ell \in \mathcal{T}, \forall s_1, s'_1 \in [1, N_1], \quad \pi_1(s_1) q_\ell(s_1, s'_1) = \pi_1(s'_1) q_\ell(s'_1, s_1).$$

## 7.2 Conditional product-form and lumping of the reversed automata

In this section we present the main theorem showing the relation between the lumping of a reversed automaton and the steady-state distribution of  $M_1 \otimes M_2$ .

**Theorem 4** (Conditional product-forms). *Given the model  $M_1 \otimes M_2$ , in a feed-forward and non-blocking synchronisation. Let  $M_1^R$  be the reversed automaton of  $M_1$  and let  $\tilde{M}_1^R$  be an exact lumping of  $M_1^R$  whose clusters are  $\mathcal{S} = \{\tilde{1}, \dots, \tilde{N}_1\}$  and*

let  $\tilde{M}_1^R$  be reversible. Then, under ergodicity assumption, the following  $\tilde{N}_1$ -order product-form expression holds:

$$\pi(s_1, s_2) = \tilde{\pi}_{M_2|\tilde{M}_1^R}^R(s_2|\tilde{s}_1)\pi_1(s_1), \quad (7.3)$$

where  $\pi$  is the steady-state distribution of  $M_1 \otimes M_2$  and  $\tilde{\pi}^R$  that of  $\tilde{M}_1^R \otimes M_2$ ,  $\pi_1$  that of  $M_1$  and:

$$\tilde{\pi}_{M_2|\tilde{M}_1^R}^R(s_2|\tilde{s}_1) = \frac{\tilde{\pi}^R(\tilde{s}_1, s_2)}{\tilde{\pi}_1(\tilde{s}_1)},$$

where, since the stochastic process underlying  $\tilde{M}_1^R$  is a lumping of that underlying  $M_1^R$ , we have  $\tilde{\pi}_1(\tilde{s}_1) = \sum_{s_1 \in \tilde{s}_1} \pi_1(s_1)$ .

*Proof of Theorem 4.* The main line of the proof consists in showing that replacing  $\pi(s_1, s_2)$  with the expression (7.3) in the GBE of  $M_1 \otimes M_2$  gives an identity.

Let us start by writing down the GBE corresponding to state  $s_1$  in  $M_1$ . Remember that, since by hypothesis we have a non-blocking synchronisation, the marginal distribution of  $M_1$  is independent of the state of  $M_2$  in the cooperation of  $M_1 \otimes M_2$ , hence:

$$\begin{aligned} \pi_1(s_1) \left( \sum_{s'_1=1}^{N_1} q_1(s_1, s'_1) + \sum_{\ell=3}^T \sum_{s'_1=1}^{N_1} q_\ell(s_1, s'_1) \right) \\ = \sum_{s'_1=1}^{N_1} \pi_1(s'_1)q_1(s'_1, s_1) + \sum_{\ell=3}^T \sum_{s'_1=1}^{N_1} p_{i_1}(s'_1)q_\ell(s'_1, s_1), \end{aligned} \quad (7.4)$$

and the GBE corresponding to state  $(\tilde{s}_1, s_2)$  in  $\tilde{M}_1^R \otimes M_2$ :

$$\begin{aligned} \tilde{\pi}^R(\tilde{s}_1, s_2) \left( \sum_{\tilde{s}'_1=\bar{1}}^{\tilde{N}_1} \tilde{q}_1^R(\tilde{s}_1, \tilde{s}'_1) + \sum_{\ell=3}^T \sum_{\tilde{s}'_1=\bar{1}}^{\tilde{N}_1} \tilde{q}_\ell^R(\tilde{s}_1, \tilde{s}'_1) + \sum_{s'_2=1}^{N_2} q_2(s_2, s'_2) \right) \\ = \sum_{\tilde{s}'_1=\bar{1}}^{\tilde{N}_1} \tilde{\pi}^R(\tilde{s}'_1, s_2)\tilde{q}_1^R(\tilde{s}'_1, \tilde{s}_1) + \sum_{\ell=3}^T \sum_{\tilde{s}'_1=\bar{1}}^{\tilde{N}_1} \sum_{s'_2=1}^{N_2} \tilde{\pi}^R(\tilde{s}'_1, s'_2)\tilde{q}_\ell^R(\tilde{s}'_1, \tilde{s}_1)E_\ell(s'_2, s_2) \\ + \sum_{s'_2=1}^{N_2} \tilde{\pi}^R(\tilde{s}_1, s'_2)q_2(s'_2, s_2). \end{aligned} \quad (7.5)$$

Observe that thanks to the hypothesis of non-blocking synchronisation, the rate out of state  $(\tilde{s}_1, s_2)$  due to a synchronising type  $\ell$  is  $\sum_{\tilde{s}'_1=\bar{1}}^{\tilde{N}_1} \tilde{q}_\ell^R(\tilde{s}_1, \tilde{s}'_1)$ , whereas when considering the synchronising transition from  $(\tilde{s}'_1, s'_2)$  to  $(\tilde{s}_1, s_2)$  we must multiply the rate of  $\tilde{q}_1^R(\tilde{s}'_1, \tilde{s}_1)$  with the synchronising probability of  $M_2$ , i.e.  $E_\ell(s'_2, s_2)$ , according to the semantics of cooperations. Equations (7.4) and (7.5) uniquely identify the probability distribution  $\pi_1(s_1)$  and  $\tilde{\pi}^R(\tilde{s}_1, s_2)$  and hence are identities. Now, we prove the following two propositions that allow for some simplifications.

**Proposition 1.** *The following relation holds:*

$$\sum_{\tilde{s}'_1=\bar{1}}^{\tilde{N}_1} \tilde{q}_1^R(\tilde{s}_1, \tilde{s}'_1) + \sum_{s'_1 \in \tilde{s}_1} \frac{\pi_1(s'_1)}{\pi_1(s_1)} q_1(s'_1, s_1) = \sum_{s'_1=1}^{N_1} \frac{\pi_1(s'_1)}{\pi_1(s_1)} q_1(s'_1, s_1),$$

with  $\tilde{s}_1$  denoting the class of state  $s_1$ .

Indeed, we can write:

$$\begin{aligned} \sum_{s'_1=1}^{N_1} \frac{\pi_1(s'_1)}{\pi_1(s_1)} q_1(s'_1, s_1) &= \sum_{\substack{\tilde{s}'_1=\bar{1} \\ \tilde{s}'_1 \neq \tilde{s}_1}}^{\tilde{N}_1} \sum_{s'_1 \in \tilde{s}'_1} q_1^R(s_1, s'_1) + \sum_{s'_1 \in \tilde{s}_1} \frac{\pi_1(s'_1)}{\pi_1(s_1)} q_1(s'_1, s_1) \\ &= \sum_{\tilde{s}'_1=\bar{1}}^{\tilde{N}_1} \tilde{q}_1^R(\tilde{s}_1, \tilde{s}'_1) + \sum_{s'_1 \in \tilde{s}_1} \frac{\pi_1(s'_1)}{\pi_1(s_1)} q_1(s'_1, s_1), \end{aligned}$$

by Definitions 6 and 3 (remember that  $\tilde{q}_1^R(\tilde{s}_1, \tilde{s}_1) = 0$  by definition).

**Proposition 2.** *The following relation holds:*

$$\sum_{\ell=3}^T \sum_{\tilde{s}_1=\bar{1}}^{\tilde{N}_1} \tilde{q}_\ell^R(\tilde{s}_1, \tilde{s}'_1) = \sum_{\ell=3}^T \sum_{s'_1=1}^{N_1} \frac{\pi_1(s'_1)}{\pi_1(s_1)} q_\ell(s'_1, s_1).$$

Using again Definitions 6 and 3, we can write:

$$\sum_{\ell=3}^T \sum_{s'_1=1}^{N_1} \frac{\pi_1(s'_1)}{\pi_1(s_1)} q_\ell(s'_1, s_1) = \sum_{\ell=3}^T \sum_{\tilde{s}'_1=\bar{1}}^{\tilde{N}_1} \sum_{s_1 \in \tilde{s}_1} q_\ell^R(s_1, s'_1) = \sum_{\ell=3}^T \sum_{\tilde{s}'_1=\bar{1}}^{\tilde{N}_1} \tilde{q}_\ell^R(\tilde{s}_1, \tilde{s}'_1).$$

Using Propositions 1 and 2, we can sum Equation (7.4) divided by  $\pi_1(s_1)$  and Equation (7.5) divided by  $\tilde{\pi}^R(\tilde{s}_1, s_2)$ , obtaining the following identity:

$$\begin{aligned} &\sum_{s'_1=1}^{N_1} q_1(s_1, s'_1) + \sum_{\ell=3}^T \sum_{s'_1=1}^{N_1} q_\ell(s_1, s'_1) + \sum_{s'_2=1}^{N_2} q_2(s'_2, s_2) \\ &= \underbrace{\sum_{\tilde{s}'_1=\bar{1}}^{\tilde{N}_1} \frac{\tilde{\pi}^R(\tilde{s}'_1, s_2)}{\tilde{\pi}^R(\tilde{s}_1, s_2)} \tilde{q}_1^R(\tilde{s}'_1, \tilde{s}_1) + \sum_{s'_1 \in \tilde{s}_1} \frac{\pi_1(s'_1)}{\pi_1(s_1)} q_1(s'_1, s_1)}_A \\ &+ \underbrace{\sum_{\ell=3}^T \sum_{\tilde{s}'_1=\bar{1}}^{\tilde{N}_1} \sum_{s'_2=1}^{N_2} \frac{\tilde{\pi}^R(\tilde{s}'_1, s'_2)}{\tilde{\pi}^R(\tilde{s}_1, s_2)} \tilde{q}_\ell^R(\tilde{s}'_1, \tilde{s}_1) E_\ell(s'_2, s_2)}_B + \underbrace{\sum_{s'_2=1}^{N_2} \frac{\tilde{\pi}^R(\tilde{s}_1, s'_2)}{\tilde{\pi}^R(\tilde{s}_1, s_2)} q_2(s'_2, s_2)}_C. \quad (7.6) \end{aligned}$$

The proof proceeds by writing the GBE corresponding to a generic state  $(s_1, s_2)$  of  $M_1 \otimes M_2$  and showing that Identity (7.6) is obtained after some algebraic manipulations by substituting the product-form expression of Equation (7.3).

The GBE for state  $(s_1, s_2)$  in  $M_1 \otimes M_2$  is:

$$\begin{aligned} \pi(s_1, s_2) & \left( \sum_{s'_1=1}^{N_1} q_1(s_1, s'_1) + \sum_{\ell=3}^T \sum_{s'_1=1}^{N_1} q_\ell(s_1, s'_1) + \sum_{s'_2=1}^{N_2} q_2(s_2, s'_2) \right) \\ & = \underbrace{\sum_{s'_1=1}^{N_1} \pi(s'_1, s_2) q_1(s'_1, s_1)}_D + \underbrace{\sum_{s'_2=1}^{N_2} \pi(s_1, s'_2) q_2(s'_2, s_2)}_E + \\ & \quad \underbrace{\sum_{\ell=3}^T \sum_{s'_1=1}^{N_1} \sum_{s'_2=1}^{N_2} \pi(s'_1, s_2) q_\ell(s'_1, s_1) E_\ell(s'_2, s_2)}_F \quad (7.7) \end{aligned}$$

Let us divide the equation by  $\pi(s_1, s_2)$  and consider only the right-hand side. After the substitution, block  $D$  can be rewritten as:

$$\begin{aligned} \frac{D}{\pi(s_1, s_2)} & = \sum_{\substack{\tilde{s}'_1=\bar{1} \\ \tilde{s}'_1 \neq \tilde{s}_1}}^{\tilde{N}_1} \sum_{s'_1 \in \tilde{s}'_1} \frac{\tilde{\pi}_{M_2|\tilde{M}_1^R}(s_2|\tilde{s}'_1)\pi_1(s'_1)}{\tilde{\pi}_{M_2|\tilde{M}_1^R}(s_2|\tilde{s}_1)\pi_1(s_1)} q_1(s'_1, s_1) + \sum_{s'_1 \in \tilde{s}_1} \frac{\pi_1(s'_1)}{\pi_1(s_1)} q_1(s'_1, s_1) \\ & = \sum_{\substack{\tilde{s}'_1=\bar{1} \\ \tilde{s}'_1 \neq \tilde{s}_1}}^{\tilde{N}_1} \sum_{s'_1 \in \tilde{s}'_1} \frac{\tilde{\pi}_{M_2|\tilde{M}_1^R}(s_2|\tilde{s}'_1)}{\tilde{\pi}_{M_2|\tilde{M}_1^R}(s_2|\tilde{s}_1)} q_1^R(s_1, s'_1) + \sum_{s'_1 \in \tilde{s}_1} \frac{\pi_1(s'_1)}{\pi_1(s_1)} q_1(s'_1, s_1) \text{ by Def. 6} \\ & = \sum_{\substack{\tilde{s}'_1=\bar{1} \\ \tilde{s}'_1 \neq \tilde{s}_1}}^{\tilde{N}_1} \frac{\tilde{\pi}(\tilde{s}'_1, s_2)\tilde{\pi}_1(\tilde{s}_1)}{\tilde{\pi}(\tilde{s}_1, s_2)\tilde{\pi}_1(\tilde{s}'_1)} \tilde{q}_1^R(\tilde{s}_1, \tilde{s}'_1) + \sum_{s'_1 \in \tilde{s}_1} \frac{\pi_1(s'_1)}{\pi_1(s_1)} q_1(s'_1, s_1) \text{ by Def. 3} \\ & = \sum_{\substack{\tilde{s}'_1=\bar{1} \\ \tilde{s}'_1 \neq \tilde{s}_1}}^{\tilde{N}_1} \frac{\tilde{\pi}(\tilde{s}'_1, s_2)}{\tilde{\pi}(\tilde{s}_1, s_2)} \tilde{q}_1^R(\tilde{s}'_1, \tilde{s}_1) + \sum_{s'_1 \in \tilde{s}_1} \frac{\pi_1(s'_1)}{\pi_1(s_1)} q_1(s'_1, s_1) = A, \end{aligned}$$

where the last manipulation is a consequence of the hypothesis that  $\tilde{M}_1^R$  is reversible, and hence  $\tilde{\pi}_1(\tilde{s}_1)\tilde{q}_1^R(\tilde{s}_1, \tilde{s}'_1) = \tilde{\pi}_1(\tilde{s}'_1)\tilde{q}_1^R(\tilde{s}'_1, \tilde{s}_1)$ . Let us consider block  $E$ :

$$\begin{aligned} \frac{E}{\pi(s_1, s_2)} & = \sum_{s'_2=1}^{N_2} \frac{\tilde{\pi}_{M_2|\tilde{M}_1^R}(s'_2|\tilde{s}_1)\pi_1(s_1)}{\tilde{\pi}_{M_2|\tilde{M}_1^R}(s_2|\tilde{s}_1)\pi_1(s_1)} q_2(s'_2, s_2) \\ & = \sum_{s'_2=1}^{N_2} \frac{\tilde{\pi}(\tilde{s}_1, s'_2)\tilde{\pi}_1(\tilde{s}_1)}{\tilde{\pi}(\tilde{s}_1, s_2)\tilde{\pi}_1(\tilde{s}_1)} q_2(s'_2, s_2) = C. \end{aligned}$$

Finally, we consider block  $F$ :

$$\begin{aligned}
\frac{F}{\pi(s_1, s_2)} &= \sum_{\ell=3}^T \sum_{s'_2=1}^{N_2} E_\ell(s'_2, s_2) \sum_{\tilde{s}'_1=\bar{1}}^{\tilde{N}_1} \sum_{s'_1 \in \tilde{s}'_1} \frac{\tilde{\pi}_{M_2|\tilde{M}_1^R}^R(s'_2|\tilde{s}'_1)\pi_1(s'_1)}{\tilde{\pi}_{M_2|\tilde{M}_1^R}^R(s_2|\tilde{s}_1)\pi_1(s_1)} q_\ell(s'_1, s_1) \\
&= \sum_{\ell=3}^T \sum_{s'_2=1}^{N_2} E_\ell(s'_2, s_2) \sum_{\tilde{s}'_1=\bar{1}}^{\tilde{N}_1} \sum_{s'_1 \in \tilde{s}'_1} \frac{\tilde{\pi}_{M_2|\tilde{M}_1^R}^R(s'_2|\tilde{s}'_1)}{\tilde{\pi}_{M_2|\tilde{M}_1^R}^R(s_2|\tilde{s}_1)} q_\ell^R(s_1, s'_1) \text{ By Def. 6} \\
&= \sum_{\ell=3}^T \sum_{s'_2=1}^{N_2} E_\ell(s'_2, s_2) \sum_{\tilde{s}'_1=\bar{1}}^{\tilde{N}_1} \frac{\tilde{\pi}_{M_2|\tilde{M}_1^R}^R(s'_2|\tilde{s}'_1)}{\tilde{\pi}_{M_2|\tilde{M}_1^R}^R(s_2|\tilde{s}_1)} \tilde{q}_\ell^R(\tilde{s}_1, \tilde{s}'_1) \text{ By Def. 3} \\
&= \sum_{\ell=3}^T \sum_{s'_2=1}^{N_2} E_\ell(s'_2, s_2) \sum_{\tilde{s}'_1=\bar{1}}^{\tilde{N}_1} \frac{\tilde{\pi}^R(\tilde{s}'_1, s'_2)\tilde{\pi}_1(\tilde{s}_1)}{\tilde{\pi}^R(\tilde{s}_1, s_2)\tilde{\pi}_1(\tilde{s}'_1)} \tilde{q}_\ell^R(\tilde{s}_1, \tilde{s}'_1) \\
&= \sum_{\ell=3}^T \sum_{s'_2=1}^{N_2} E_\ell(s'_2, s_2) \sum_{\tilde{s}'_1=\bar{1}}^{\tilde{N}_1} \frac{\tilde{\pi}^R(\tilde{s}'_1, s'_2)}{\tilde{\pi}^R(\tilde{s}_1, s_2)} \tilde{q}_\ell^R(\tilde{s}'_1, \tilde{s}_1) = B,
\end{aligned}$$

where in the last passage we have used the assumption that  $\tilde{M}_1^R$  is reversible.

Concluding, by substituting the product-form expression (7.3) in Equation (7.7), we obtain the identity (7.6) and hence (7.3) is the unique normalised solution of  $M_1 \otimes M_2$ .  $\square$

**Corollary 2.** *The marginal distribution of  $M_2$  may be computed as:*

$$\pi_2(s_2) = \sum_{\tilde{s}_1=\bar{1}}^{\tilde{N}_1} \pi(\tilde{s}_1, s_2).$$

**Proof of Corollary 2.** We have to show that:

$$\sum_{\tilde{s}_1=\bar{1}}^{\tilde{N}_1} \pi(\tilde{s}_1, s_2) = \sum_{s_1=1}^{N_1} \pi(s_1, s_2).$$

We substitute in the right hand-side of this equation the product-form expression (7.3) obtaining:

$$\begin{aligned}
\sum_{s_1=1}^{N_1} \pi(s_1, s_2) &= \sum_{\tilde{s}_1=\bar{1}}^{\tilde{N}_1} \sum_{s_1 \in \tilde{s}_1} \pi(s_1) \tilde{\pi}_{M_2|\tilde{M}_1^R}^R(s_2|\tilde{s}_1) = \sum_{\tilde{s}_1=\bar{1}}^{\tilde{N}_1} \tilde{\pi}_{M_2|\tilde{M}_1^R}^R(s_2|\tilde{s}_1) \tilde{\pi}_1(\tilde{s}_1) \\
&= \sum_{\tilde{s}_1=\bar{1}}^{\tilde{N}_1} \pi(\tilde{s}_1, s_2)
\end{aligned}$$

$\square$

**Example 9** (Lumping of the reversed automaton). *Let us consider again the automaton of Figure 6.1 and its lumping depicted in Figure 6.2. Notice that this latter automaton is reversible. Therefore, if we consider  $M_1^R$  to be the automaton of Figure 6.1 and  $\tilde{M}_1^R$  that of Figure 6.2, we can apply Theorem 4 provided that the cooperating automaton  $M_2$  is non-blocking and under the ergodicity assumption. In this context, to obtain the forward automaton corresponding to  $\tilde{M}_1$  it suffices to reverse it obtaining the rates shown in Table 7.1. The steady-state probabilities of  $M_1$  (and  $M_1^R$ ) are:*

$$\begin{aligned}\pi_1(A1) &= \frac{2\gamma(2\lambda + \mu + 2\eta)}{(\epsilon + 3\gamma)(\beta + 2\lambda + 2\mu + 2\eta)} \\ \pi_1(A2) &= \frac{2\gamma(\beta + \mu)}{(\epsilon + 3\gamma)(\beta + 2\lambda + 2\mu + 2\eta)} \\ \pi_1(B) &= \frac{\gamma}{\epsilon + 3\gamma} \\ \pi_1(C1) &= \frac{\epsilon(3\delta + 4\gamma)}{3(\alpha + \delta + 2\gamma)(\epsilon + e\gamma)} \\ \pi_1(C2) &= \frac{\epsilon(3\alpha + 2\gamma)}{3(\alpha + \delta + 2\gamma)(\epsilon + e\gamma)}\end{aligned}$$

and those of  $\tilde{M}_1^R$  are:

$$\tilde{\pi}_1(A) = \frac{2\gamma}{3\gamma + \epsilon}, \quad \tilde{\pi}_1(B) = \frac{\gamma}{3\gamma + \epsilon}, \quad \tilde{\pi}_1(C) = \frac{\epsilon}{3\gamma + \epsilon},$$

where  $A = \{A1, A2\}$  and  $C = \{C1, C2\}$ . Therefore, for any automaton  $M_2$  such that all the matrices  $M_{2\ell}$ , with  $\ell \neq 2$ , are stochastic, we have that:

$$\pi(s_1, s_2) = \pi_1(s_1) \frac{\tilde{\pi}(s_1, s_2)}{\tilde{\pi}_1(s_1)}.$$

### 7.2.1 Theoretical considerations about Theorem 3 and 4

In this section we have presented two theorems about lumping in cooperating stochastic models. We already pointed out the connections between Theorem 3 and the notion of strong equivalence (and its consequences) presented in [101]. Now, we compare Theorem 4 with other relatively recent results that appeared in literature, in particular for what concerns the analysis of product-form stochastic models. A model is in product-form if the joint stationary distribution of an ergodic joint state can be expressed in terms of the product of the marginal distributions of its components considered in isolation and opportunely parametrised. A very general theory about product-form models has been developed in [89] where the author, based on

Action Type $\ell$	Incoming state	Outgoing state	$\lambda_\ell \mathbf{E}_{1\ell}$
1	$B$	$A_1$	$\frac{2\mu(2\lambda+\mu+2\eta)}{(\beta+2\lambda+2\mu+2\eta)}$
1	$A_1$	$A_2$	$\frac{\eta(\beta+\mu)}{2\lambda+\mu+2\eta}$
1	$B$	$A_2$	$\frac{2\mu(\beta+\mu)}{(\beta+2\lambda+2\mu+2\eta)}$
1	$A_1$	$B$	$\frac{\mu(\beta+2\lambda+2\mu+2\eta)}{2(2\lambda+\mu+2\eta)}$
1	$A_2$	$B$	$\frac{\mu(\beta+2\lambda+2\mu+2\eta)}{2(\beta+\mu)}$
3	$B$	$A_1$	$\frac{2\lambda(2\lambda+\mu+2\eta)}{\beta+2\lambda+2\mu+2\eta}$
3	$B$	$A_2$	$\frac{2\lambda(\beta+\mu)}{\beta+2\lambda+2\mu+2\eta}$
3	$A_1$	$B$	$2\frac{\lambda(\beta+2\lambda+2\mu+2\eta)}{2(2\lambda+\mu+2\eta)}$
3	$C_2$	$C_1$	$\frac{\alpha(3\delta+4\gamma)}{3\alpha+2\gamma}$
3	$C_1$	$C_2$	$\frac{\gamma(3\alpha+2\gamma)}{(3\delta+4\gamma)}$
4	$A_1$	$A_1$	$\beta/2$
4	$A_2$	$A_1$	$\frac{\beta(2\lambda+\mu+2\eta)}{2(\beta+\mu)}$
4	$A_2$	$A_2$	$\beta$
4	$C_1$	$B$	$\frac{\gamma(\alpha+\delta+2\gamma)}{(3\delta+4\gamma)}$
4	$C_2$	$B$	$\frac{2\gamma(\alpha+\delta+2\gamma)}{(3\alpha+2\gamma)}$
4	$B$	$C_1$	$\frac{\epsilon(3\delta+4\gamma)}{3(\alpha+\delta+2\gamma)}$
4	$C_1$	$C_1$	$\delta$
4	$B$	$C_2$	$\frac{\epsilon(3\alpha+2\gamma)}{3(\alpha+\delta+2\gamma)}$
4	$C_1$	$C_2$	$\frac{\delta(3\alpha+2\gamma)}{(3\delta+4\gamma)}$

Table 7.1: Rates of the reversed automaton corresponding to the automaton of Figure 6.1.

process algebra analysis, gives sufficient conditions for the cooperation of two models to be in product-form (Reversed Compound Agent Theorem -RCAT-). If we reformulate those conditions in terms on cooperating stochastic automata, we have:

- C1** If transition  $t > 2$  is passive with respect to  $M_k$ , then each state of  $M_k$  has exactly one outgoing transition of type  $t$  (and its weight is 1);
- C2** If transition  $t > 2$  is active with respect to  $M_k$ , then each state of  $M_k$  has exactly one incoming transition of type  $t$ ;
- C3** Let  $t > 2$  be an active type with respect to  $M_k$ , then the reversed rate associated with each transition of type  $t$  in  $M_k$  is the same.

Observe that Conditions C2 and C3 imply that the timed-reversed automaton  $M_1^R$  associated with  $M_1$  admits a lumping of one single state as illustrated by the following example. From this we observe that when Theorem 4 is applied and an automaton can be lumped into a single state, then we can also say that the joint steady-state probability is given by the product of the marginal distributions of the single automata.

**Example 10** (G-network analysis). *G-networks [78] are very powerful and versatile class of models developed in queueing theory and they can be efficiently studied because they yield a product-form stationary distribution. Let us consider the model of Figure arrive at Q1 and Q2 according to independent Poisson processes with rates  $\lambda_1$  and  $\lambda_2$ , respectively. Service time is exponentially distributed with mean  $\mu_1^{-1}$  and  $\mu_2^{-1}$  for Q1 and Q2, respectively. At a service completion epoch at Q1, the customer can move to Q2 as an ordinary customer with probability  $p_{12}^+$ , while it can enter Q2 as a negative customer with probability  $p_{12}^-$ . The effect of a negative customer arrival at Q2 is to delete a positive one if the queue is non-empty (otherwise the negative customer simply vanishes). In [89] it is shown that the reversed rates of the transitions with types 3 and 4 in Q1 are constant and equal to  $\lambda_1 p_{12}^+$  and  $\lambda_1 p_{12}^-$ , respectively. According to RCAT we can obtain the marginal distribution of Q2 by setting the rates of transitions with type 3 to  $\lambda_1 p_{12}^+$  and those of transitions with type 4 to  $\lambda_1 p_{12}^-$ . Analogously, the application of Theorem 4 leads to a lumping of  $M_1^R$  consisting of one single state with two self-loops: one with type 3 and rate  $\lambda_1 p_{12}^+$  and the other with type 4 and rate  $\lambda_1 p_{12}^-$ .*

Does the product-form property yield in case of a lumping to a single state applying Theorem 3? The answer is negative as shown by the following counterexample.

**Example 11.** *This example is intentionally very simple in order to spot the difference between performing a lumping in the forward process or in the reversed. We consider two exponential queues with synchronised arrivals modelled by a Poisson process with rate  $\lambda$  as shown in Figure 7.2. Service rates are  $\mu_1$  and  $\mu_2$ . In this case we can apply Theorem 3, and queue Q1 is lumped into a model  $\tilde{M}_1$  with a single*



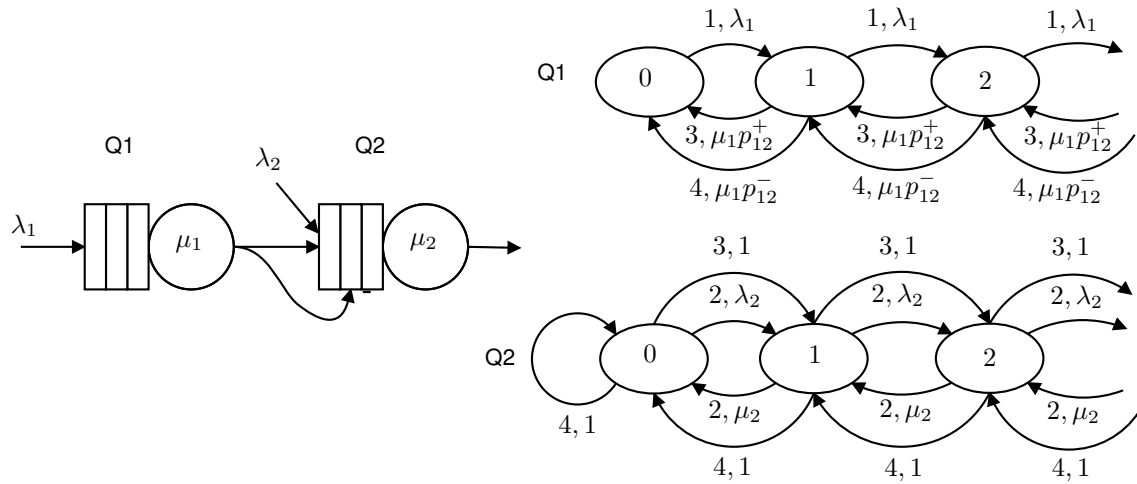


Figure 7.1: Graphical representation of a G-network and the corresponding model using automata.

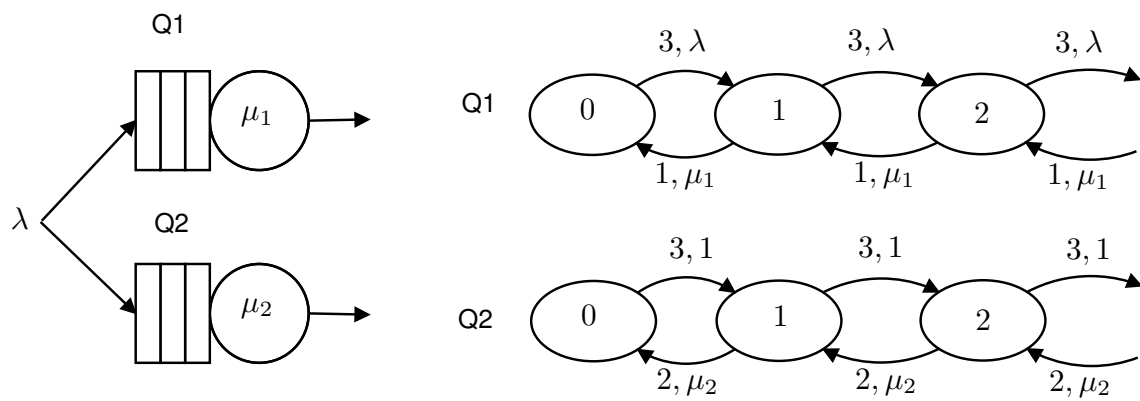


Figure 7.2: Exponential queues with synchronised arrivals and their representation by stochastic automata.

state with one self-loop with type 3 and rate  $\lambda$ . Observe that although the marginal distribution obtained for  $Q_2$  by the analysis of  $\tilde{M}_1 \otimes Q_2$  is trivially correct, the model is not in product-form since the stationary probability is different from the product of the marginal distributions of  $Q_1$  and  $Q_2$  (although they can be derived without the derivation of the joint state space).

### 7.3 Conclusions

In this and in the previous chapter we have addressed the problem of lumping stochastic automaton. We have shown a result on lumping both for forward-time automaton and reversed-time. In the latter case we show a connection between the expression of the joint steady-state probabilities and conditional product-forms. Component-wise lumping may generate a joint model with a higher number of states than that obtained by the optimal lumping applied straightforwardly on the joint CTMC [2]. However, the former approach never requires to define the whole joint model and hence in many case may tackle the state space explosion problem. Algorithmically, the problem of lumping Markov chains, PEPA models, queueing networks and stochastic Petri nets has been widely addressed. Here, we mention the important algorithm based on a notion of isomorphism among PEPA components presented in [83]. The isomorphism relation is stricter than the strong equivalence (which is itself stricter than the conditions of Definition 3) and hence, as the same authors point out, the resulting lumping may be suboptimal. However, the algorithm is very efficient, already implemented in the PEPA Workbench [81] and the results it provides may be straightforwardly used to apply Theorem 3. The same algorithm may still be applied once the time-reversed automaton has been derived and hence apply Theorem 4.

For what concerns future developments, we are currently working on proving that some stochastic models which arise from practical applications exhibits the kind of conditional product-form that we have introduced in this chapter.

---

# Approximate aggregation techniques

## 8.1 Introduction

As we have seen in the previous chapter, component-wise states space aggregation can lead to more efficient solutions of cooperating stochastic models. However, it is not always the case that a model exhibits a lumpable structure. Moreover, even when this is the case, finding the right clustering, i.e., a partition of the state space, in order to apply the aforementioned results is not a trivial task.

In this context we mentioned the algorithm presented in [83], which is based on a notion of isomorphism among PEPA components. The isomorphism relation is stricter than the strong equivalence (which is itself stricter than the conditions of Definition 3) and hence, as the same authors point out, the resulting lumping may be not optimal. However, the algorithm is very efficient, already implemented in the PEPA Workbench [81] and the results it provides may be straightforwardly used to apply Theorem 3. The same algorithm may still be applied once the time-reversed automaton has been derived and hence apply Theorem 4. Note that the problem of time-reversing PEPA agents has already been widely addressed in [89].

In this chapter we aim at exploiting the results stated in chapters 6 and 7 to define an algorithm that computes an approximation of the stationary distribution of the passive model  $M_2$  in the cooperation  $M_1 \otimes M_2$ .

## 8.2 Evaluation of the quality of clusters

In this section we address the problem of measuring how close an arbitrary state partition is to an exact lumping as given in Definition 3. Let  $M_1$  be the active model and  $\mathcal{T}$  the transition type set with the convention that  $t = 1$  ( $t = 2$ ) denotes the type of the transitions that  $M_1$  ( $M_2$ ) can carry out independently of  $M_2$  ( $M_1$ ), and  $t > 2$  denotes the type for the synchronised transitions in which  $M_1$  is active. Given an arbitrary partition  $\mathcal{W}$  (note that we reserve  $\mathcal{S}$  for denoting partitions that are also lumpings), we measure the coefficient of variation of the outgoing fluxes  $\phi_1^t(s_1)$  of the states in  $\tilde{s}_1$ . In the following definitions we assume the empty cluster (that would lead to a 0/0 in the definition) to have error 0.

**Definition 8** ( $\epsilon$ -error). *Given model  $M_1$  and a partition of states  $\mathcal{W} = \{\tilde{1}, \dots, \tilde{N}_1\}$ , for all  $\tilde{s}_1 \in \mathcal{W}$  and  $t > 2$ , we define:*

$$\begin{aligned}\bar{\phi}_1^t(\tilde{s}_1) &= \left( \sum_{s_1 \in \tilde{s}_1} \pi_1(s_1) \phi_1^t(s_1) \right) \frac{1}{\sum_{s_1 \in \tilde{s}_1} \pi_1(s_1)} \\ \epsilon^t(\tilde{s}_1) &= 1 - \exp \left( - \sqrt{ \frac{\sum_{s_1 \in \tilde{s}_1} \pi_1(s_1) (\phi_1^t(s_1) - \bar{\phi}_1^t(\tilde{s}_1))^2}{\sum_{s \in \tilde{s}_1} \pi_1(s)} } \right).\end{aligned}$$

Observe that  $0 \leq \epsilon^t(\tilde{s}_1) < 1$  and if automaton  $\tilde{M}_1$  is an exact lumping of  $M_1$ , then  $\forall t > 2, \forall \tilde{s}_1 = 1, \dots, \tilde{N}_1$  we have that  $\epsilon^t(\tilde{s}_1) = 0$ . We use the minimisation of error  $\epsilon$  to perform a first rough clustering of the states of  $M_1$  as described in Section 8.3. The following definition gives a more accurate measure of the error of a partition.

**Definition 9** ( $\delta$ -error). *Given model  $M_1$  and a partition of states  $\mathcal{W} = \{\tilde{1}, \dots, \tilde{N}_1\}$ , for all  $\tilde{s}_1, \tilde{s}'_1 \in \mathcal{W}$ , we define:*

$$\begin{aligned}\bar{\varphi}_1^t(\tilde{s}_1, \tilde{s}'_1) &= \begin{cases} 0 & \tilde{s}_1 = \tilde{s}'_1 \wedge t = 1 \\ \frac{(\sum_{s_1 \in \tilde{s}_1} \pi_1(s_1) \varphi_1^t(s_1, \tilde{s}'_1))}{\sum_{s_1 \in \tilde{s}_1} \pi_1(s_1)} & \text{otherwise} \end{cases} \\ (\sigma^t(\tilde{s}_1, \tilde{s}'_1))^2 &= \sum_{s_1 \in \tilde{s}_1} \frac{\pi_1(s_1) (\varphi_1^t(s_1, \tilde{s}'_1) - \bar{\varphi}_1^t(\tilde{s}_1, \tilde{s}'_1))^2}{\sum_{s \in \tilde{s}_1} \pi_1(s)} \\ \delta^t(\tilde{s}_1, \tilde{s}'_1) &= 1 - e^{-\sigma(\tilde{s}_1, \tilde{s}'_1)}\end{aligned} \tag{8.1}$$

where function  $\varphi_1^t$  has been defined in Definition 3.

Similarly to  $\epsilon^t$ , also for error  $\delta^t$  we have that  $0 \leq \delta^t(\tilde{s}_1, \tilde{s}_2) < 1$ .

### 8.3 Algorithm definition

In this section we propose an algorithm to approximate the marginal distributions of cooperating stochastic automata. Informally, the algorithm exploits the heuristics defined for solving clustering problems in order to obtain an automaton  $M_1^{\approx}$ , which is close to an exact lumping of  $M_1$ , where we use the  $\delta$ - and  $\epsilon$ -errors to measure the goodness of the approximation of  $M_1^{\approx}$ . In what follows we focus our attention on clustering  $M_1$ , but the same discussion is obviously valid for  $M_1^R$ . An *ideal* algorithm based on the theory we developed should work as illustrated in Table 8.1. The idea is that the analyst specifies the models and a pair of tolerance constants,  $\epsilon \geq 0$ , and  $\delta \geq 0$  and the algorithm uses the best approximated lumping (i.e., that with the smallest number of clusters) for which each synchronising transition type satisfies the

conditions stated in Steps 1 and 3 to compute the marginal steady-state distribution of the components. The algorithm surely terminates because when the number of clusters of the partition is equal to the number of states of  $M_1$ , we have an exact lumping. Obviously, if one desires to specify the number of clusters of  $M_1^{\approx}$  rather

<ul style="list-style-type: none"> <li>• <b>Input:</b> automata <math>M_1, M_2, \mathcal{T}</math>, tolerances <math>\epsilon \geq 0, \delta \geq 0</math></li> <li>• <b>Output:</b> marginal distribution <math>\pi_1</math> of <math>M_1</math>; approximated marginal distribution of <math>M_2</math></li> </ul> <ol style="list-style-type: none"> <li>1. Find the minimum <math>\tilde{N}'_1</math> such that there exists a partition <math>\mathcal{W} = \{\tilde{1}, \dots, \tilde{N}'_1\}</math> of the states of <math>M_1</math> such that <math>\forall t \in \mathcal{T}, t &gt; 2</math> and <math>\forall \tilde{s}_1 \in \mathcal{W} \epsilon(\tilde{s}_1) \leq \epsilon</math></li> <li>2. Let <math>\mathcal{W}' \leftarrow \mathcal{W}</math></li> <li>3. Check if partition <math>\mathcal{W}'</math> is such that <math>\forall t \in \mathcal{T}, \forall \tilde{s}_1, \tilde{s}_2 \in \mathcal{W}, \tilde{s}_1 \neq \tilde{s}_2, \delta^t(\tilde{s}_1, \tilde{s}'_1) \leq \delta</math>. If this is true then return the marginal distribution of <math>M_1</math> and the approximated of <math>M_2</math> by computing the marginal distribution of <math>\tilde{M}_1 \otimes M_2</math> and terminate.</li> <li>4. Otherwise, refine partition <math>\mathcal{W}</math> to obtain <math>\mathcal{W}^{new}</math> such that the number of clusters of <math>\mathcal{W}^{new}</math> is greater than the number of clusters in <math>\mathcal{W}'</math>. <math>\mathcal{W}' \leftarrow \mathcal{W}^{new}</math>. Repeat from Step 3</li> </ol>
---

Table 8.1: Ideal algorithm for computing the approximated marginal distributions of cooperating automata.

than the tolerance criteria, the algorithm can be simplified. The following definition specifies how we derive an approximate lumped automaton given a partition of its states  $\mathcal{W}$ .

**Definition 10** (Approx. lumped automata). *Given active automaton  $M_1$ , a set of transition types  $\mathcal{T}$ , and a partition of the states of  $M_1$  into  $\tilde{N}_1$  clusters  $\mathcal{W} = \{\tilde{1}, \tilde{2}, \dots, \tilde{N}_1\}$ , then we define the automaton  $M_1^{\approx}$  as follows:*

$$\tilde{\mathbf{E}}_{11}(\tilde{s}_1, \tilde{s}'_1) = \begin{cases} \overline{\varphi}_1^1(\tilde{s}_1, \tilde{s}'_1) \tilde{\lambda}_1^{-1} & \text{if } \tilde{s}_1 \neq \tilde{s}_2 \\ 0 & \text{otherwise} \end{cases} \quad \tilde{\mathbf{E}}_{12} = \mathbf{I}, \tilde{\mathbf{E}}_{1t}(\tilde{s}_1, \tilde{s}_1) = \overline{\varphi}_1^t(\tilde{s}_1, \tilde{s}'_1) \tilde{\lambda}_t^{-1} \quad t > 2$$

where  $\tilde{\lambda}_t = \max_{\tilde{s}_1=1, \dots, \tilde{N}_1} \left( \sum_{\tilde{s}'_1=1}^{\tilde{N}_1} \overline{\varphi}_1^t(\tilde{s}_1, \tilde{s}'_1) \right)$  are the rates associated with the transition types in the cooperation between  $M_1^{\approx}$  and  $M_2$ .

The algorithm is called *ideal* because the problem of performing an optimal clustering is known to be NP-hard and hence a sub-optimal solution is usually computed using some heuristics [176]. We now consider the main steps of the algorithm

of Table 8.1 and discuss how they can be implemented in practice. Note that the choice of an algorithm often depends on the characteristics of the dataset that one is studying. Since we consider general automata, we decided to propose at least two different solutions for each of the problems proposed by the algorithm of Table 8.1, i.e., implementing Step 1 and Step 4.

### The initial clustering based on $\epsilon$ -error

The initial clustering can be implemented with various algorithms. The similarity measure between two states  $s_1$  and  $s'_1$  can be the Euclidean distance between the vectors  $(\phi_1^3(s_1), \dots, \phi_1^T(s_1))$  and  $(\phi_1^3(s'_1), \dots, \phi_1^T(s'_1))$ . In case of hierarchical clustering a divisive (top-down) approach can be adopted and the algorithm stops when the condition of  $\epsilon$ -error stated in Step 1 is satisfied. Another approach, which is used in the example we propose in Section 8.4, is to use K-means. K-means is a fast algorithm for clustering whose drawback consists in the need of specifying a priori the desired number of clusters  $K$ . Therefore, this is usually the good choice if some intuition can drive the analyst in choosing an initial value for  $K$ . If the constraints on  $\epsilon$  cannot be satisfied by the choice of  $K$ , then K-means must be run again specifying a number of clusters  $K' > K$ .

### Refining the clustering obtained in Step 1

We consider the problem of refining the clustering obtained in Step 1 in order to satisfy the conditions required by the tolerance constant  $\delta$ . Note that often the partitions obtained from the first step are sufficient to obtain good approximations in real models such as queues (as shown by the Example of Section 8.4). Let  $\mathcal{W} = \{\tilde{s}_1, \dots, \tilde{s}_n\}$  be the clustering obtained by the first phase of the algorithm and we want to obtain  $\mathcal{W}' = \{\tilde{p}_1, \dots, \tilde{p}_m\}$  with  $m > n$  such that  $\mathcal{W}'$  is a refinement of  $\mathcal{W}$ . Note that K-means cannot be straightforwardly applied to cluster the states because the distance measures among the states depend on the cluster themselves.

**Spectral analysis** Spectral analysis is applied to stochastic matrices  $\mathbf{P}$ , therefore we must discretise the CTMC underlying automaton  $M_1$ . Let  $\mathbf{Q}_1$  be the infinitesimal generator of  $M_1$ , then, following [165], let  $\Delta t = \max(|\mathbf{Q}(i, i)|)$ , where  $\mathbf{Q}(i, i)$  are the diagonal elements of  $\mathbf{Q}$ . Then, we have  $\mathbf{P} = \mathbf{Q}\Delta t + \mathbf{I}$ . Now observe that an exact lumping for  $\mathbf{P}$  is also an exact lumping for  $\mathbf{Q}$ , and a good approximation for  $\mathbf{P}$  is also a good approximation for  $\mathbf{Q}$ . However, recall that an exact lumping of CTMC corresponding to  $\mathbf{Q}$  is a necessary condition for satisfying Definition 3 but not sufficient. This method has been introduced by Jacobi in [108] for identifying approximated lumpings in Markov chains. The algorithm relies on the property that a Markov chain admits an exact lumping with  $K$  states if and only if there are exactly  $K$  right eigenvectors of  $\mathbf{P}$  with elements that are constant over the aggregates (see [108] and the references therein), thus it defines  $\mathbf{R} = \mathbf{P}^\top \mathbf{P} - \mathbf{P} - \mathbf{P}^\top + \mathbf{I}$  and computes

its eigenvalues. Then, it obtains the eigenvectors  $\mathbf{u}_i$  corresponding to the  $K$  smallest eigenvalues. Finally, the algorithm associates a vector  $\mathbf{v}_j$  with each state  $j$  of the process that is defined as  $\mathbf{v}_j = (u_j(1), \dots, u_j(K))$ . The last step consists in running K-Means to obtain  $K$  clusters. Note that the original algorithm uses the Euclidean distance between vectors to perform the clustering, however we need to obtain a partition  $\mathcal{W}'$  that is a refinement of  $\mathcal{W}$ . Therefore, we define the distance function  $d$  as:

$$d(\mathbf{v}_i, \mathbf{v}_j) = \begin{cases} \|\mathbf{v}_i - \mathbf{v}_j\|_2 & \text{if } i, j \in \tilde{s}_1 \text{ for some } s_1 \in \mathcal{W} \\ \top & \text{otherwise} \end{cases}.$$

**Iterative algorithm** This algorithm is an adaptation to our framework of the class of clustering algorithms described in [120]. We briefly describe the original version as illustrated in [108]. Let  $\mathbf{Z}$  be a  $N_1 \times K$  matrix whose elements are in  $\{0, 1\}$  and each row has exactly one non-zero element and  $K \leq N_1$ . Let  $\tilde{\mathbf{P}}$  be a  $K \times K$  matrix. Then, if we can find a matrix  $\mathbf{Z}$  such that  $\mathbf{P}\mathbf{Z} = \mathbf{Z}\tilde{\mathbf{P}}$ ,  $\tilde{\mathbf{P}}$  denotes the transition matrix of a lumped process. Elements  $\mathbf{Z}(s_1, \tilde{s}_1)$  of  $\mathbf{Z}$  are 1 if  $s_1 \in \tilde{s}_1$ , 0 otherwise, with  $1 \leq s_1 \leq N_1$  and  $1 \leq \tilde{s}_1 \leq K$ . The approximate lumping is obtained by iteratively assigning a random state  $s_1$  to cluster  $\tilde{s}'_1$ , where  $\tilde{s}'_1$  is chosen as the cluster that minimises the local error measure  $\|(\mathbf{P}\mathbf{Z})(s_1, *) - \mathbf{P}^\approx(\tilde{s}'_1, *)\|_2$ , where  $\mathbf{P}^\approx$  is a candidate partition and  $\mathbf{P}^\approx(s, *)$  denotes row  $s$  of the matrix. Obviously, the algorithm can converge to a local minimum. For a comparison between this approach and the spectral decomposition see [120, 108].

## 8.4 Example

In this section we consider an example consisting of a model in which the active automaton is not exactly lumpable, and, by applying the approach described above, we show how state aggregation of an automaton can both drastically reduce the state space cardinality and provide a good approximation of the marginal steady-state probabilities of the passive automaton. Consider the queueing network represented in Figure 8.1 where the stations Q1 and Q2 have a finite capacity  $C1$  and  $C2$  and customers arrive at Q1 and Q2 according to homogeneous Poisson processes of parameter  $\lambda_1$  and  $\lambda_2$ , respectively. The behaviour of the customers arriving at the first queue can be described by a function  $\gamma : \{0, \dots, C1\} \rightarrow [0, \lambda_1]$  that given  $n_1$ , i.e., the number of customers already present in Q1, gives the rate of customers skipping the first queue, while the rate of the customers that regularly enqueue in the first station is given by  $\lambda_1 - \gamma(n_1)$ , where function  $\gamma$  is defined as follows:

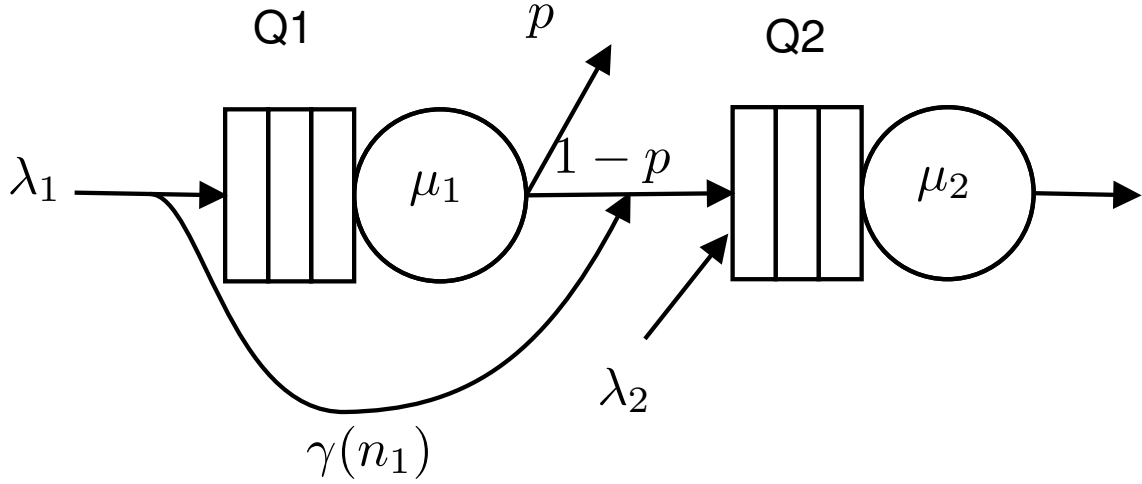


Figure 8.1: Example model.

$$\gamma(n_1) = \begin{cases} 0 & \text{if } n_1 \leq \left\lfloor \frac{C1}{2} \right\rfloor \\ \frac{\lambda_1}{2} & \text{if } \left\lfloor \frac{C1}{2} \right\rfloor < n_1 < C1 \\ \lambda_1 & \text{if } n_1 = C1 \end{cases}$$

When Q2 is full, customers are lost. Figure 8.2 shows the component-wise stochastic processes underlying the described system, where  $P1$  and  $P2$  are the stochastic automata for Q1 and Q2, respectively and state  $h$  is the first one where  $\gamma$  assumes the value  $\lambda_1/2$ . We shall apply the techniques and algorithms proposed in Section 8.3 to this example and we present a comparison of our results with those obtained by other methods. The parameters are  $C1 = 20, C2 = 20, \lambda_1 = 6, \lambda_2 = 1, \mu_1 = 4, \mu_2 = 4, p = 0.7$ . Using an implementation of the ideal algorithm of Table 8.1 on the process  $P1$  and enforcing the use of  $K$  clusters, one should get a partition  $\mathcal{L} = \{L_1, \dots, L_K\}$ , where each  $L_i$  is a subset of the states of  $P1$ . Using our implementation with  $\epsilon = 10^{-13}$  and  $\delta = 0.95$  as tolerances, as we were interested in evaluating a coarse-grained approximated lumping, we obtained 4 clusters, i.e.,  $L_1 = \{0\}, L_2 = \{1, \dots, 10\}, L_3 = \{11, \dots, 19\}$  and  $L_4 = \{20\}$ . Notice that the resulting clustering is not an exact lumping, thus Theorem 3 does not hold. We can however use this clustering to compute an approximation of the marginal steady-state probabilities of the process  $P2$ , reducing drastically the state space of the joint process. Using the same technique, we partitioned the reversed process  $P1^R$  of  $P1$ , obtaining  $\bar{L}_1 = \{0, \dots, 10\}, \bar{L}_2 = \{11, 12\}, \bar{L}_3 = \{13, \dots, 19\}$  and  $\bar{L}_4 = \{20\}$ , which is not an exact lumping, thus Theorem 4 cannot be applied.

In order to evaluate the results of our technique, we compared them with the solution obtained with other ones, namely we obtained the marginal steady state probabilities of  $P2$  using the following methods:



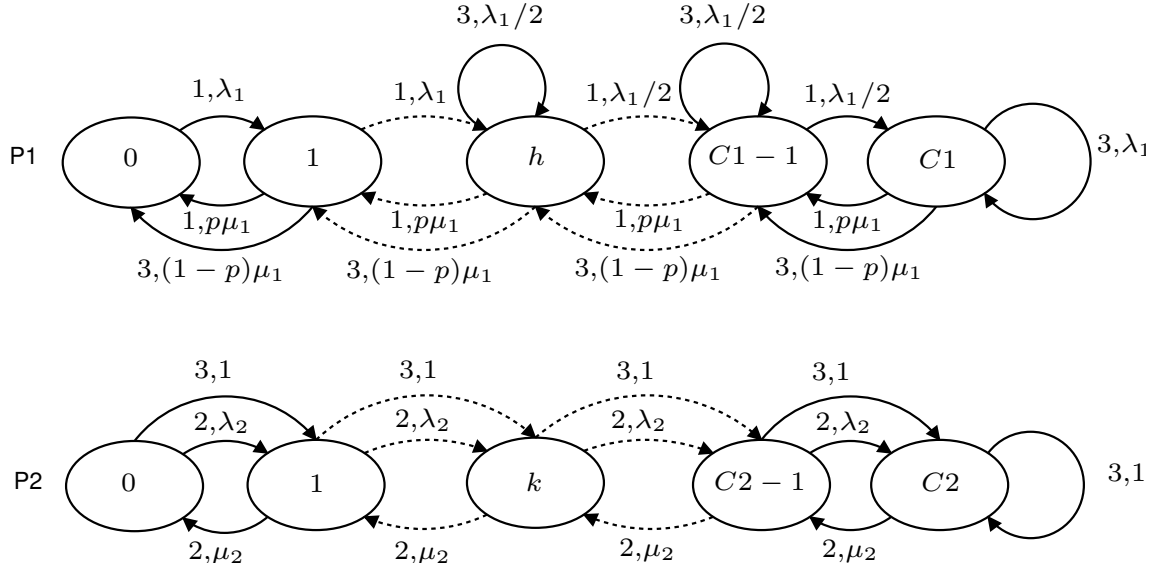


Figure 8.2: Automata of the example model.

1. The computation of the joint probabilities between the approximate lumping  $\mathcal{L}$  of  $P1$  and the process  $P2$ . [FW-Lump]
2. The computation of the joint probabilities between the approximate lumping  $\bar{\mathcal{L}}$  of the of the *reversed* process  $P1^R$  and the process  $P2$ . [RV-Lump]
3. The Approximated Product Form of order 4 introduced in [46]. [APF]
4. The Fixed Point Approximation [140]. [FPA]
5. The exact computation of the joint probabilities between  $P1$  and  $P2$ . [Exact]

In order to compare the quality of the distributions obtained by the analysed methods, we used the *Kullback-Leibler divergence* between the exact marginal probability distribution of  $P2$  and these approximations, computed as:

$$D_{KL}(P||Q) = \sum_i P(i) \log \frac{P(i)}{Q(i)},$$

where  $P$  is the exact probability distribution and  $Q$  is the estimated one. Table 8.2 reports the Kullback-Leibler divergence, the average number of customers  $E[N]$  and its relative error for  $Q2$ . While all the methods offer a reasonable approximation on average performance indices, the error can be remarkable on their steady-state distribution. Although we cannot consider these numerical results a complete study of the relative merit of the various methods, it is possible to point out some differences among them. In particular, in this case, the results given by the approximate lumping of the reversed process are more accurate than those obtained using the

	FW-Lump	RV-Lump	APF	FPA	Exact
KL div.	0.0065	0.0045	0.0451	0.0112	0
$E[N]$	11.62	11.55	9.990	11.80	11.33
Rel. err.	0.0259	0.0200	0.1178	0.0424	0

Table 8.2: Comparison between approximation methods.

forward process. This is because grouping states by the sum of outgoing rates of synchronising transition types leads to 4 sets in the forward process, and to 3 sets in the reversed one. This allows the algorithm to refine one more time one of the clusters by an enforced split in 4 components.

## 8.5 Conclusions

In this chapter we have proposed a methodology for approximating the marginal distributions in the cooperations of two stochastic processes through approximated lumping using results from Theorems 3 and 4. The main contribution relies in the evaluation of lumping quality through  $\epsilon$  and  $\delta$ -error, and in the possibility to choose the better lumping between the one on the forward and on the one on reversed process. The advantages of this approach lie on the fact that it is done on single cooperating automata and that, under some assumptions, it can lead to better approximations with respect to other popular techniques, as shown in section 8.4. Future research directions include the application of our methodology to real case studies with large state spaces and the investigation on the relations between error metrics and clustering algorithm.

# III

---

## Case studies and applications



---

# 9

## Exploiting product forms solution techniques in multiformalism modelling

### 9.1 Introduction

Computer based systems that daily serve most human activities are characterized by an increasing level of complexity, a term that can be interpreted in different ways. Complexity can be found in the concurrent requirements these systems have to satisfy while functioning (e.g. for critical systems, that have temporal, safety, dependability and performance constraints), from the number of their different articulations, or from their extension. Furthermore, complexity can be the result of a substanding logic of design by composition of existing subsystems. In all these cases, designers face the challenge by exploiting models to define or understand the system's characteristics.

Multiformalism modelling techniques allow to choose the most suitable formalism to consider different aspects or parts of the overall system, thus providing a more manageable and understandable model. These techniques support a modular approach, in which components are designed separately. However, a component-based approach does not imply an efficient model solution.

As we have seen in Part II, while standard model analysis techniques suffer from the so-called state space explosion problem, in some cases it is possible to exploit the compositional definition of the system in order to find efficient solution methods.

In this chapter, based on a work we published in [22], we show how the product-form solution theory and techniques described in 5 easily couple with multiformalism compositional modelling techniques, to obtain a modelling and analysis framework that offers modelling flexibility and efficient solutions. The contribution is based on the design and implementation of an extensible modelling and solution framework, supported by a tool solving multiformalism Markovian models with a threefold solution mechanism. The tool automatically verifies and performs a product-form solution. If this is not available it provides a state space based analytical solution or a simulation as final backup tool. The research extends the SIMTHESys frame-

work and the tool for product-form solutions, presented in Chapter 5, in order to encompass product form models that satisfy the ERCAT [91] and MARCAT [92] theorems. To date, it appears there is no other similar, tool-supported approach in the current literature.

A review on existing multi-formalism tools and approaches can be found in Chapter 4

## 9.2 From multiformalism models to product-form solutions

The approach proposed in this chapter leverages the existing modularity in multiformalism models and the possibilities of the SIMTHESys [106] framework to exploit Markov chains product-form solutions detection in a multiformalism solution process. The SIMTHESys framework supports the design and development of user-defined formalisms, for which a proper (multi)formalism solver is automatically generated. The framework defines a formalisms design technique, based on metamodelling and on a mechanism to specify the structural and dynamic characteristics of every element of a formalism [25, 24, 23]. Solver generation is based on the analysis of the formalism description and on a set of basic *solving engines*. In this chapter, the Markov chain solving engines (analytical and simulative) are used, together with a new solving engine that benefits from the INAP algorithm and the MARCAT Theorem.

The set of specifications that a (multi)formalism should satisfy to apply MARCAT to the model will be presented by developing proper high level formalisms, whose specifications are inspired by Plateau's SAN [146]. We introduce a new *formalism family*, that is a set of features that a formalism should have in order to allow models expressed using it to be solved using the product-form computational engine. The set of features that characterize this family of formalisms is inspired by the features available in Continuous Time Stochastic Automata Networks with Master/Slave synchronisations [63]. This new formalism family will be called *Labelled Exponential Events Formalisms*. Formalisms belonging to this family can be conveniently thought as Labelled Exponential Automata.

The approach is implemented by using SIMTHESysER, the SIMTHESys framework solver generation tool, and the tool described in chapter 5, extended in order to check the requirements of the MARCAT theorem. Within this approach, the proposed formalism family allows the SIMTHESys framework to generate product-form solution based optimized solvers and the tool to support high level formalisms, extending its application field to more complex models. Moreover, a further result of the integration of the two tools is the enrichment of the benefits of MARCAT with the automatic verification of its hypotheses by using SIMTHESysER state space generation logic. This provides a tool that allows the identification of a greater

number of known product-forms without the need of checking them one by one.

The main limit of the approach is that it is only applicable to models that cooperate pairwise. As a consequence more complex product-forms such as those based on instantaneous signal propagation [79] are not considered. When product-forms are not applicable, models are solved by the generated solver using the general analytic approach or by simulation.

### 9.2.1 Deciding and computing the product-form solution

In this section we outline the main steps that are performed to decide if a model admits a MARCAT based product-form solution and, in case of positive answer, to compute it.

A model belonging to the *Labelled Exponential Events Formalism* considered in this work can be considered as a tuple  $(C, L)$ , where  $C = \{c_1 \dots c_N\}$  is the set of sub-model components, and  $L = \{l_1 \dots l_K\}$  is a set of labels. Sub-models can be defined in any formalism that belongs to the considered family. In particular, each sub-model component  $c_i$  is characterized by a set of variables  $V_i = \{v_{i,1} \dots v_{i,n_i}\}$  that define its state, and a set of events  $E_i = \{e_{i,1} \dots e_{i,m_i}\}$  that governs the transition from one state to another. Each state  $\mathcal{S}_i$  is uniquely identified by the value of its variables: if two states have the same values for all the variables of the model, they are the same state. The information contained in a state  $\mathcal{S}_i$  is capable of completely defining the possible events  $e_i \in E_i$  that can cause a state change, and their temporal behaviour. Events that triggers a change of state can occur either locally after an exponentially distributed time, or globally due to a synchronization. Exponential events  $e_i$  are characterized by a rate  $\lambda(e_i, \mathcal{S}_i) \rightarrow \mathcal{R}^+$ : as soon as the system enters state  $\mathcal{S}_i$ , event  $e_i$  will occur after an exponentially distributed random amount of time, with rate  $\lambda(e_i, \mathcal{S}_i)$ . If more than one event can occur in the same state  $\mathcal{S}_i$ , then *race policy* is used to choose between the two. Synchronization is performed following a *Master/Slave* paradigm over a label. *Active events* have an exponential time associated and a label:  $\mu(e_i, \mathcal{S}_i) \rightarrow \mathcal{R}^+ \times L$ . *Passive events* have only a label associated with them:  $\gamma(e_i, \mathcal{S}_i) \rightarrow L$ . When an *Active event* is available in a state, it is triggered after the corresponding exponentially distributed random amount of time. During the execution of an Active event, the associated label  $l_j$  is generated. Passive events are instead immediately executed as soon as the corresponding label  $l_j$  is generated by an active event in another sub-model: this allows synchronization among the sub-models. All the events moves the system into another state, by appropriately changing the values of the variables that defines the states of the various sub-models involved by the events (that is, of the considered sub-model for local events, or of the sub-models where the Active and Passive events belong), that is  $\mathcal{S}'_i = f(e_i, \mathcal{S}_i) \quad \forall s_i \in S$ .

In order to keep the chapter self-contained we briefly present MARCAT in case of a pair of cooperating models  $P \equiv c_1$  and  $Q \equiv c_2$  (in this case the theorem is usually known as ERCAT). Henceforth we denote by  $P \otimes Q$  as the joint model.

Assume that for each label  $a$  we know a positive real value  $x_a$ , and let  $P'$  ( $Q'$ ) be the process  $P$  ( $Q$ ) in which all the passive transitions labelled by  $a \in \mathcal{P}_P$  ( $a \in \mathcal{P}_Q$ ) take  $x_a$  as a rate. Clearly,  $P'$  and  $Q'$  are now independent and, if the underlying CTMCs are ergodic (or have an ergodic subset of states) we can compute their steady-state distributions  $\pi_{P'}$  and  $\pi_{Q'}$ . According to MARCAT, a product-form solution exists if for each ergodic state of  $(p, q)$  of  $P \otimes Q$  we have that its steady-state probability  $\pi(p, q)$  is:  $\pi(p, q) \propto \pi_{P'}(p)\pi_{Q'}(q)$ , where the direct proportionality symbol is an equality if the ergodic states of  $P \otimes Q$  are the Cartesian product of the ergodic states of  $P'$  and  $Q'$ . The following sets play a pivotal role in the application of MARCAT, we recall that  $(p, q)$  is an ergodic state of  $P \otimes Q$ :

1.  $\mathcal{P}^{(p,q)\rightarrow}$  is the set of passive labels outgoing from  $(p, q)$
2.  $\mathcal{A}^{(p,q)\rightarrow}$  is the set of active labels outgoing from  $(p, q)$
3.  $\mathcal{P}^{(p,q)\leftarrow}$  is the set of passive labels entering into  $(p, q)$
4.  $\mathcal{A}^{(p,q)\leftarrow}$  is the set of active labels entering into  $(p, q)$

MARCAT gives a purely algorithmic way to decide the existence of a product-form solution in the cooperation of two processes, and in case of existence it states its expression.

**Theorem 5** (MARCAT [92]). *Let  $P$  and  $Q$  be two cooperating processes on set of labels  $\mathcal{L}$  and assume that the following conditions are satisfied:*

- For each label  $a \in \mathcal{A}_P$  ( $a \in \mathcal{A}_Q$ ),  $x_a$  is the reversed rate of all the transitions labelled by  $a$  in  $P'$  ( $Q'$ );
- For each joint state  $(p, q)$  the following rate equation holds:

$$\begin{aligned} \sum_{a \in \mathcal{P}^{(p,q)\rightarrow}} x_a - \sum_{a \in \mathcal{A}^{(p,q)\leftarrow}} x_a \\ = \sum_{a \in (\mathcal{P}^{(p,q)\leftarrow} \setminus \mathcal{A}^{(p,q)\leftarrow})} \bar{\beta}_a(p, q) - \sum_{a \in (\mathcal{A}^{(p,q)\rightarrow} \setminus \mathcal{P}^{(p,q)\rightarrow})} \alpha_a(p, q) \end{aligned} \quad (9.1)$$

where  $\alpha_a(p, q)$  is the rate of the transition outgoing from state  $(p, q)$  labelled by  $a$  while  $\bar{\beta}_a^{(p,q)}$  is the reversed rate of the transition entering into  $(p, q)$  labelled by  $a$ ;

then for each ergodic state  $(p, q)$  of the joint process the following relation holds:  $\pi(p, q) \propto \pi_{P'}(p)\pi_{Q'}(q)$

In practice, the application of MARCAT requires to address the following problems:



- Determining the values  $x_a$  for each synchronising label;
- Checking Equation (9.1) for each ergodic state of the joint model. In particular the computation of the values  $\bar{\beta}_a$  requires the knowledge of the steady-state distribution of the component in which  $a$  is passive;

Here, the first problem is solved by applying INAP+, assuming that a product-form exists and we check Equation (9.1) *a posteriori*. INAP+ output contains both the value of the reversed rates and the steady-state distribution of each component in isolation. This information is therefore used to verify MARCAT rate condition (9.1) within a given numerical precision.

### 9.2.2 The formalisms

SIMTHESys is a flexible tool that allows for the definition of various modelling formalisms according to the users' needs and expertise. Here, we present only the two formalisms that are used in the case-study of Section 9.3. The first is a variant of open, finite capacity, blocking, repetitive services queueing networks, as already seen; the second is a variant of stochastic Petri nets: both the formalisms are enriched by including elements that implement the given specification. The elements of both formalisms are in Fig. 9.1.

For what concerns queueing networks, besides the *queue* element, characterized by its capacity, its length and its service rate, the queueing network formalism has six other node elements: the *source* element generates requests and is characterized by its rate; the *active source* element generates requests at its rate and exports a label; the *passive source* element generates requests at a rate depending on the bound label; the *sink* element consumes requests; the *active sink* element consumes requests and exports a label; the *passive sink* element consumes requests depending on the bound label. The formalism also has the *arc* element (that routes requests between sources, queues and sinks), the *test arc* element (that checks a condition over a node element to enable another node element) and the *inhibitor arc* element (that inhibits a node element according to the state of another node element).

The state of the sub-models is simply defined by the occupancy of the queues. The local events are used to account for ends of service in a queue, and for an arrival to the system. Active events are used to consider departures from the system from sinks that have an associated label (active sinks), and arrivals to the system coming from an active source. Passive events are used to model both passive arrivals and passive departures: the former corresponds to arrivals in the system triggered by the firing of an active transition belonging to a different sub-model. The latter to customer immediately leaving a system due to an interrupt coming from a different source. Moreover, queues, sources and sinks can be blocked when the total queue length of neighbour stations (but always in the same sub-mode) become larger or smaller than a given threshold thanks to *Test* and *Inhibitor Arcs*.

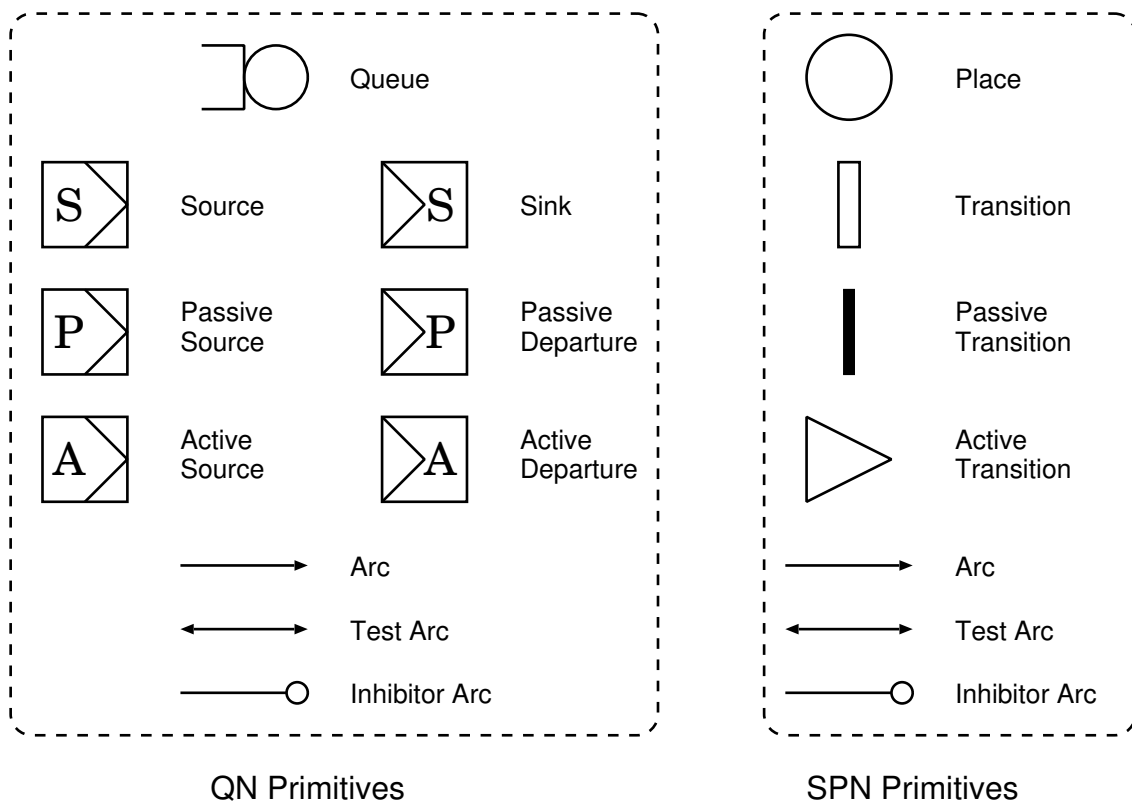


Figure 9.1: Formalisms elements

The queueing network semantic is implemented in this way. First enabled events are determined by looking at the size of the queue of the various stations. All *Source* and *Active Source* primitives always generate an event (a *local* event for the former, and an *active* event for the latter). *Passive Sources* generate *passive* events. *Queue* primitives generate events if the corresponding queue has at least one customer. The event is *local* if the destination of the queue is another queue or a *Sink*, it is *active* if the destination is an *Active Departure*, and *passive* for *Passive Departures*.

The stochastic Petri nets formalism has four node elements: the *Place* element, characterized by its marking; the *Transition* element, characterized by its rate; the *Active Transition* element, characterized by its rate and the label it exports; the *Passive Transition* element, the rate of which depends on the bound label. The formalism has arc elements analogously to the queueing network formalism. *Active Transitions* behave exactly as standard SPNs timed transition, but they also expose a label when they fire. *Passive Transitions* instead, are completely governed by the *active* events happening in other sub-models. Petri Nets are implemented by generating an event for each enabled transition. In particular, the type of the event generated corresponds to the type of transition: *local* events for standard *Transitions*, *active* events for *Active Transitions* and *passive* events for *Passive Transitions*.

Note that in our approach labels, active and passive events, are used to provide a formalism independent cooperation scheme that allows the synchronization among sub-models, specified using different modelling languages. The sub-models are connected using cooperation arcs in the enclosing main model. Each cooperation arc has associated a set of labels and it is directed from a source sub-model to a destination sub-model: whenever an active events, with the corresponding label, happens in the source sub-model, it triggers enabled passive events, associated with the same label, in the destination sub-model.

## 9.3 Case study

In this section we introduce an example that is analysed with the proposed tool. Notice that the system has been chosen to spot the original features of the technique described in this chapter.

A data stream processing system for the detection and monitoring of seismic phenomena is structured in two main subsystem: a pre-processing subsystem and a critical detection subsystem. The first is composed of two stages, each of which processes batches with a temporally variable computation, whose duration is exponentially distributed. Both the stages receive different jobs to be processed, with an exponentially distributed rate. In addition, the second receives part of the output batches of the first and some of the jobs processed by the second are sent back to be processed by the first, while the others constitute the output of the subsystem. Each of the stages can buffer a number of timestamped jobs. A stage that is not allowed to dispatch a job to the next one has to reprocess the data, to account for

the time elapsed.

The second subsystem processes the output jobs of the first one, together with additional jobs that are sent to it. To protect this critical subsystem, a protection mechanism can shut it temporarily off when the number of jobs is greater than a fixed threshold and an overload condition is detected. In such cases, the arrival of new jobs is blocked and losses occur. The protection mechanism is supplied by a proper subsystem.

### 9.3.1 Overall model description

A high-level, conceptual model of the system is given in Fig. 9.2. The first subsystem can be described by a two nodes queueing network with finite capacity and repetitive service blocking of the type studied in [16]. The second subsystem can be described by a finite capacity queue controlled by an ON/OFF switching element when the number of jobs in the queue is not less than the threshold value  $m$ . In the figure,  $\lambda_1$ ,  $\lambda_2$  and  $\lambda_3$  denote the job arrival rates;  $B_1$ ,  $B_2$  and  $B_3$  are the capacity of the queues;  $\mu_1$ ,  $\mu_2$  and  $\mu_3$  are the job service rates;  $p$  is the probability of forwarding a processed job from the first to the second queue;  $q$  is the probability of forwarding a processed batch from the second to the first queue;  $n$  is the number of jobs in the third queue;  $\gamma$  represents the trigger by which ON/OFF blocks forwarding from the first to the second subsystem, given that  $n \geq m$ .

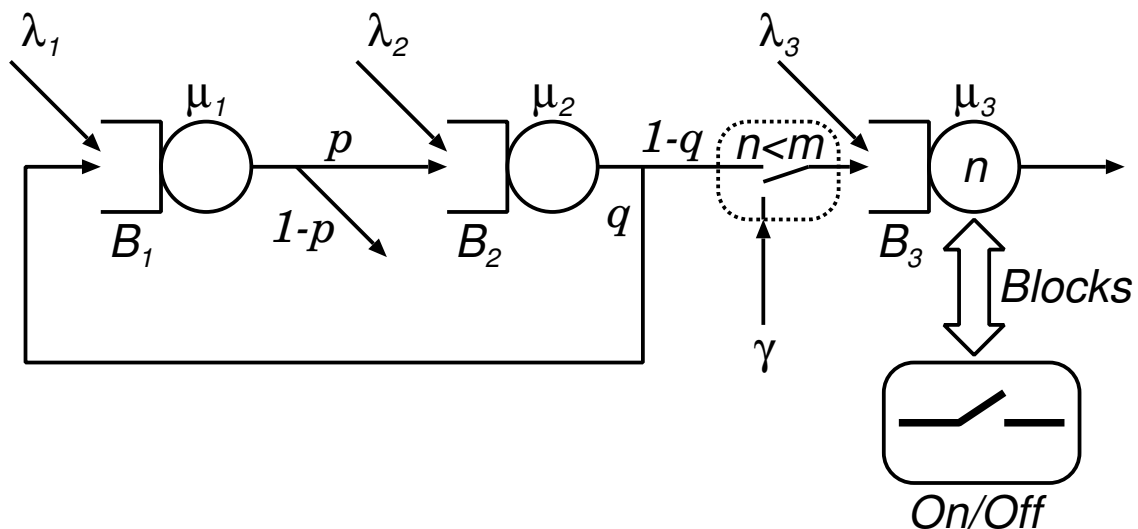


Figure 9.2: Overall model description

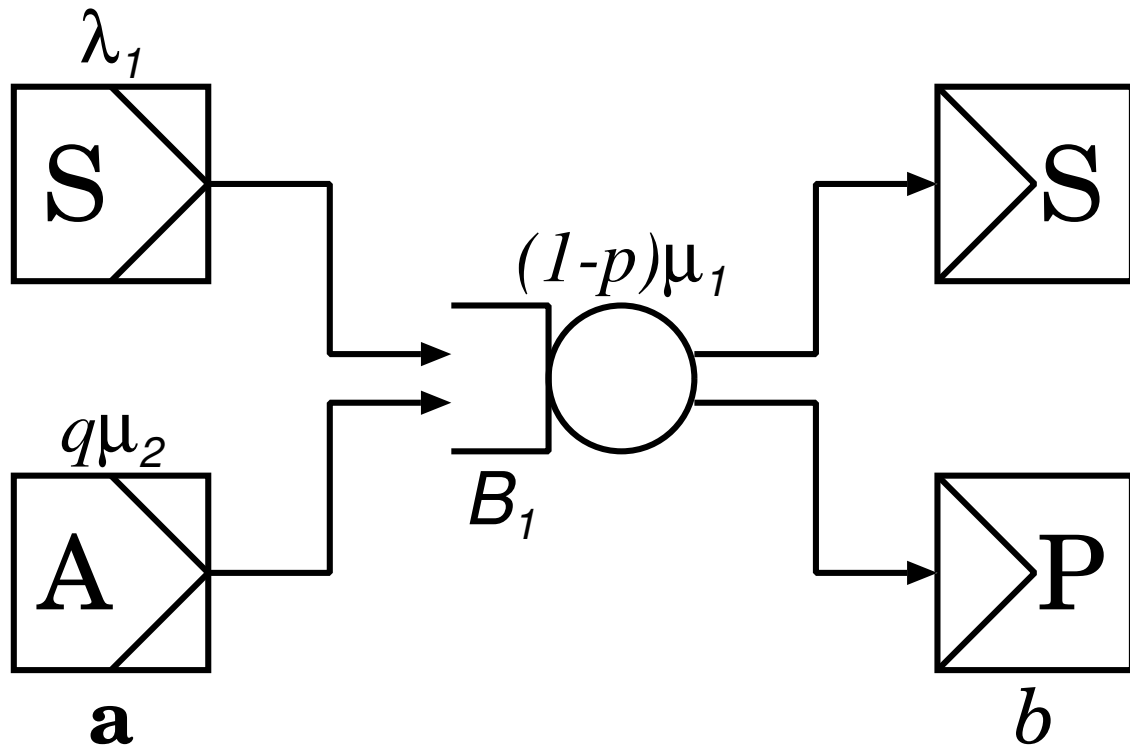


Figure 9.3: Submodel QN1

### 9.3.2 Model specification

The first subsystem is modelled by a queueing network with finite capacity and repetitive service blocking. With reference to Fig. 9.2, the first stage is described by submodel QN1 in Fig. 9.3. The source with rate  $\lambda_1$  represents the external arrivals, the active source with rate  $q\mu_2$  and exporting label  $a$  represents the jobs arriving from the second stage ( $\mu_2$  is the second stage's service rate and  $q$  the probability that a job enters the first stage after being served at the second). The sink represents the jobs that leave the system, the passive departure importing label  $b$  represents the jobs available for entering the second stage, and the queue with capacity  $B_1$  and rate  $(1-p)\mu_1$  represents the processing unit. The queue rate expression results from the fact that the only rate that can be defined here is the rate of departures by the sink, because the rate of departures for the second stage depends on the behaviour of the second stage (see [16]).

The second stage is described by submodel QN2 in fig. 9.4. Similarly as seen for the first stage, the source with rate  $\lambda_2$  represents the external arrivals, the active source with rate  $p\mu_1$  and exporting label  $b$  represents the jobs arriving from the first stage, the sink represents the jobs that leave the system, the passive departure importing label  $a$  represents the jobs available for the first stage, and the queue with

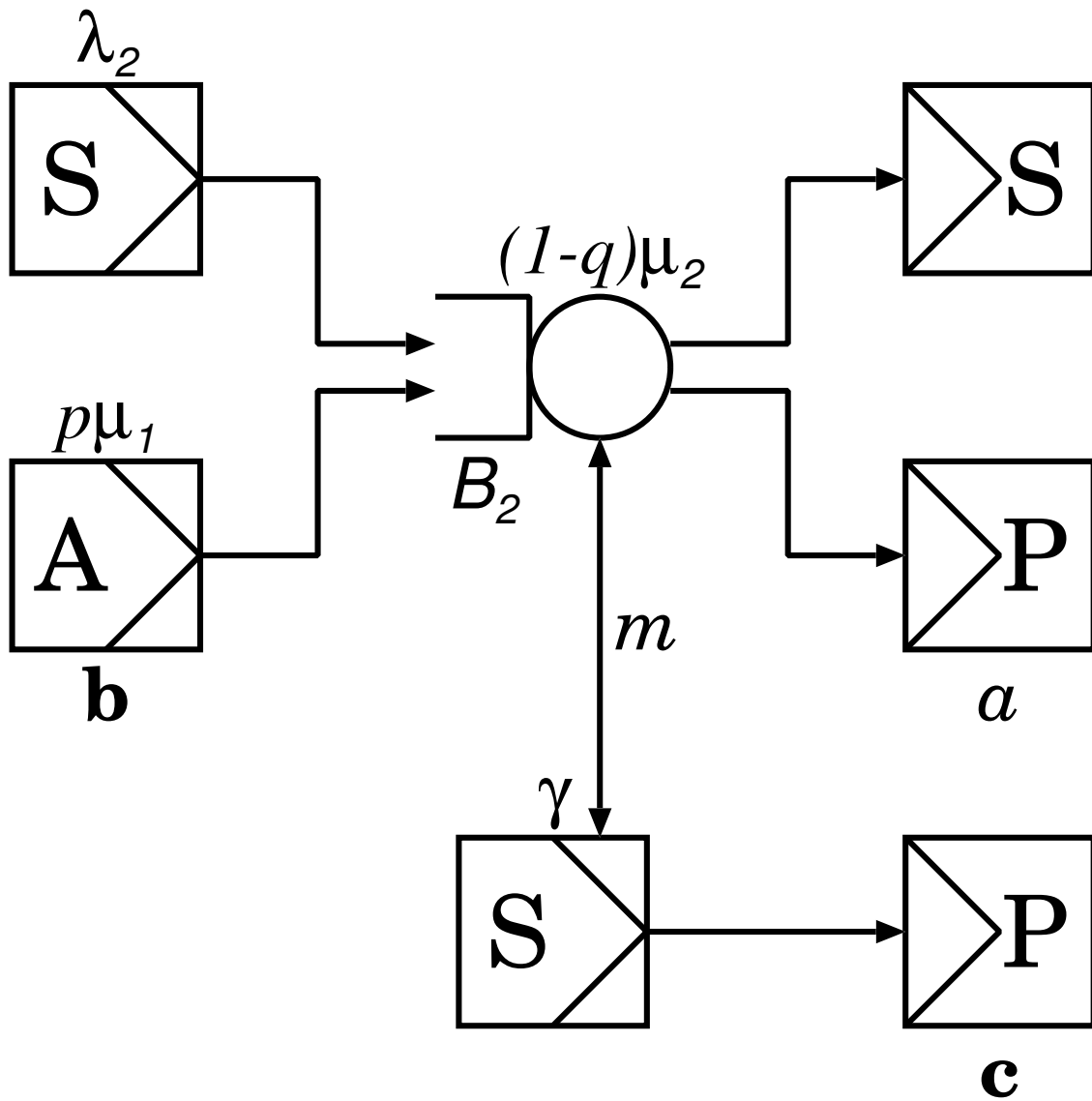


Figure 9.4: Submodel QN2

capacity  $B_2$  and rate  $(1 - q)\mu_2$  represents the processing unit. Moreover, the second stage also has a source with rate  $\gamma$ , enabled by the queue by the test arc that checks if it contains less than  $m$  pending requests, and a passive departure, that imports the label  $c$ .

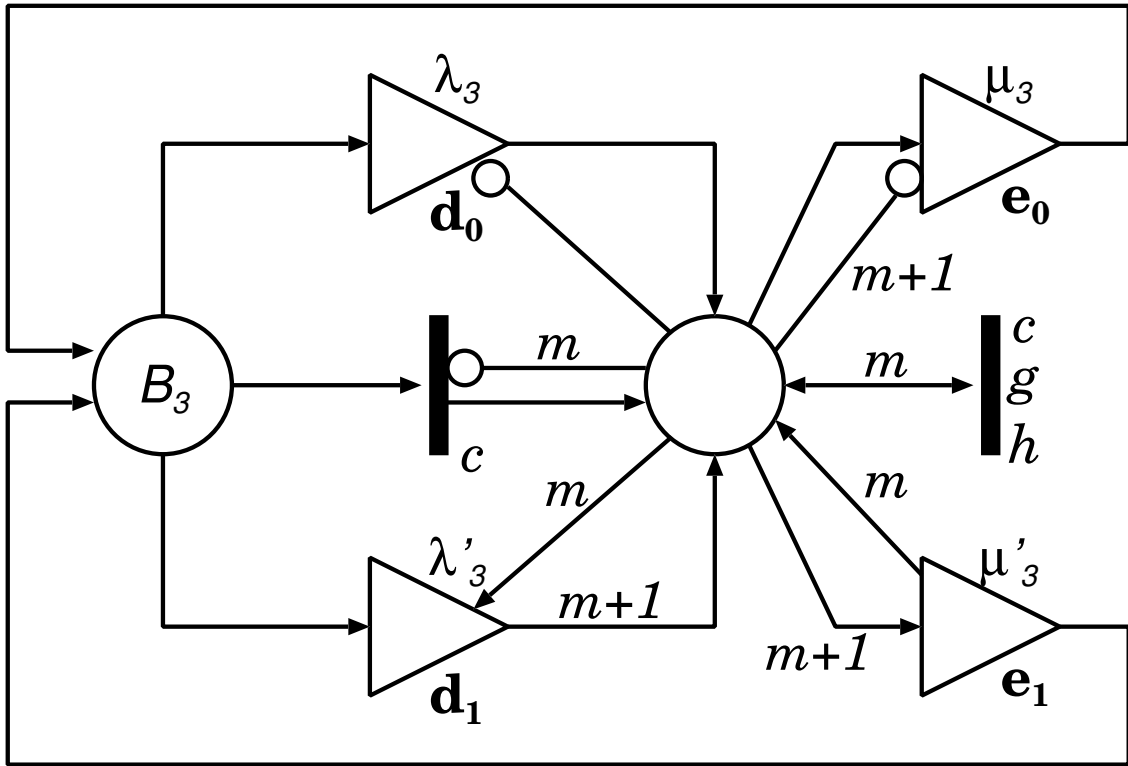
The second subsystem is modelled by two stochastic Petri net submodels. The third queue in Fig. 9.2 is described by submodel SPN1 in Fig. 9.5. The active transition with rate  $\lambda_3$  and exporting label  $d_0$  represents the external arrivals, and is enabled if the left place (initially marked with  $B_3$  tokens to represent the finite capacity) is marked and if the right place (marked by the same active transition and representing a request under processing) is not marked. The active transition with rate  $\lambda'_3$  and exporting label  $d_1$  represents the external arrivals when the length of the queue is at least  $m$ , marks the right place, and is enabled if the left place is marked and if the right place has at least  $m$  tokens (thus the queue length is at least  $m$ ); the passive transition importing label  $c$  represents arrivals from the first subsystem, and is enabled when the left place is enabled and the right place is marked with less than  $m$  tokens (thus the queue length is less than  $m$ ); the active transition with rate  $\mu_3$  and exporting label  $e_0$  represents the processing of a request when the length of the queue is less than  $m + 1$ . The active transition with rate  $\mu'_3$  and exporting label  $e_1$  represents the processing of a request when the length of the queue is at least  $m + 1$ . The passive transition importing labels  $c$ ,  $g$  and  $h$  authorizes incoming requests from the first subsystem and the ON/OFF mechanism when the length of the queue is at least  $m$ .

The ON/OFF submodel is described by two places, representing the ON (left) and the OFF (right) conditions. The active transitions with rate  $\delta$  and exporting label  $h$  and with rate  $\epsilon$  and exporting label  $g$  respectively represent the switch off and the switch on; the passive transition importing labels  $d_0$ ,  $d_1$ ,  $e_0$  and  $e_1$  enables the interactions with the previous submodel when the left place is marked.

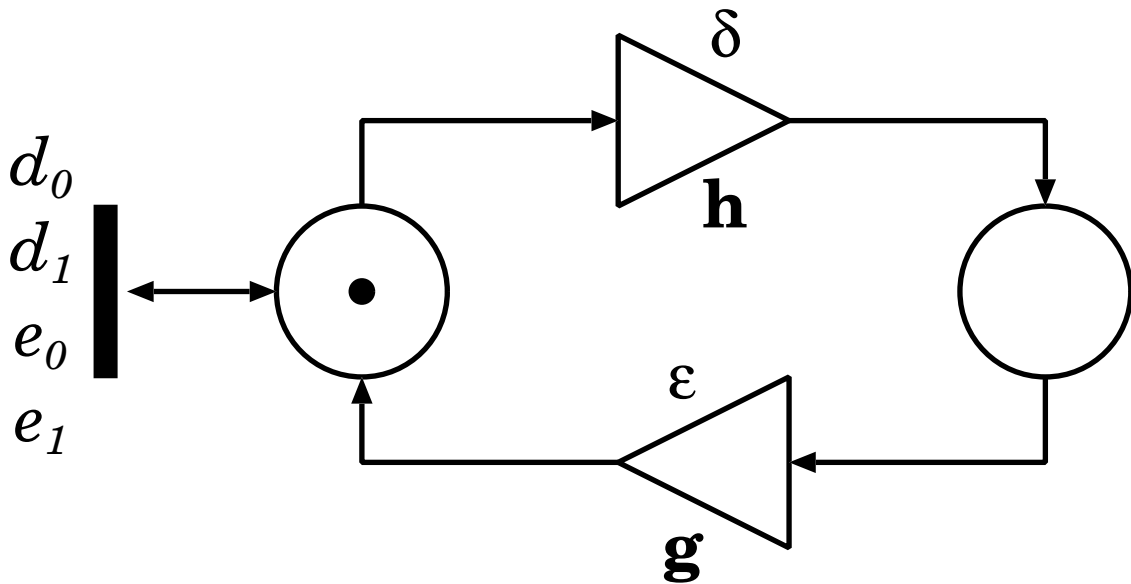
Fig. 9.6 shows how submodels are connected by the bridge model. In particular, each submodel is drawn as a rectangle, and cooperation between models is denoted by arcs. Arrows are directed from submodels performing active transitions, to submodels subject to passive events. Active event labels are written in boldface near an OUT keyword, while passive event labels in italic near an IN keyword.

### 9.3.3 Model analysis and results

The output produced by SIMTHESysER for the model considered in this example is shown in Fig. 9.7 and is expressed in terms of Labelled Exponential Automata where symbol  $\top$  denotes passive transitions. Notice that a brute-force analysis relying on the solution of the system of global balance equation could be unfeasible. Indeed, although finite, the cardinality of the state space grows as  $\mathcal{O}(B_1 B_2 B_3)$ , where  $B_i$  is the capacity of the  $i$ -th queue. Assuming for simplicity that  $B_i = B$ , the standard solution of the global balance equation system with a Gauss elimination equivalent method has a time complexity of  $\mathcal{O}(B^9)$  which could quickly become punitive also



(a) SPN1



(b) SPN2

Figure 9.5: SPN submodels



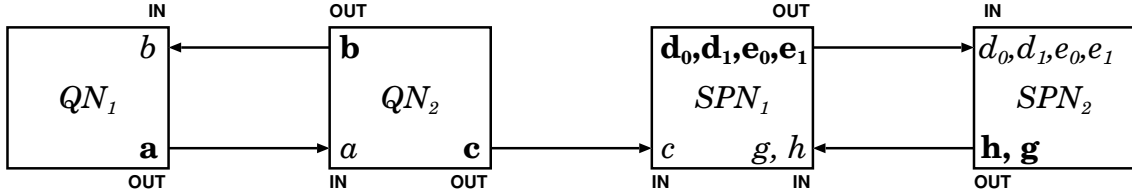


Figure 9.6: Submodels composition

for the numerical stability of the involved algorithms. Simulation could be another approach for estimating the model’s performance measures. However, we should note that the precision of the simulation estimates strongly depends on the model’s parameters. In fact, some events are likely to be rare such as the saturation of the queues or the blocking mechanism implemented for the system protection. As a consequence, time expensive simulations could be required and the validation of the estimates play an important role.

Applying product-form analysis to this model is not trivial. Indeed, none of submodels involved in the cooperation is quasi-reversible, and therefore the triviality of the derivation of the steady-state distribution is avoided. The queueing network with feedback consisting of models  $QN1$  and  $QN2$  does not admit a Jackson’s product-form since the stations have finite capacity. Nevertheless, their rate-dependent product-form relies on the analysis carried out in [16]. The output process of  $QN2$  feeds  $SPN1$ ; this process combined with the external arrivals at  $SPN1$  satisfies RCAT condition. Finally, the process that regulates the interaction between the blocking protection mechanism ( $SPN2$ ) and  $SPN1$  belongs to the Boucherie’s product-form model class [39]. The combination of these types of product-forms have a non-trivial solution given by the following proposition.

**Proposition 3.** *The case-study model has a product-form solution if the following rate-conditions are satisfied:*

$$(1 - p)q\lambda_2 = (1 - q)p\lambda_1 \tag{9.2}$$

$$\gamma = \frac{\lambda_1(1 - q)(\lambda_2 + p\mu_1)}{\lambda_1(1 - q) + (1 - p)q\mu_1} \tag{9.3}$$

*Proof.* It can be algebraically proved that the set of equations (9.1) are satisfied if and only if Condition (9.2) and (9.3) are satisfied.  $\square$

The product-form analytical solution of the case-study model depends on the choice of some rates as illustrated by Proposition 3. Although this is somehow disappointing from a modelling point of view, the importance is not only theoretical. For instance, a product-form parameterisation of the model could be applied to validate the simulation results, or in case the rate conditions are not satisfied, an approximated analysis could be carried out.

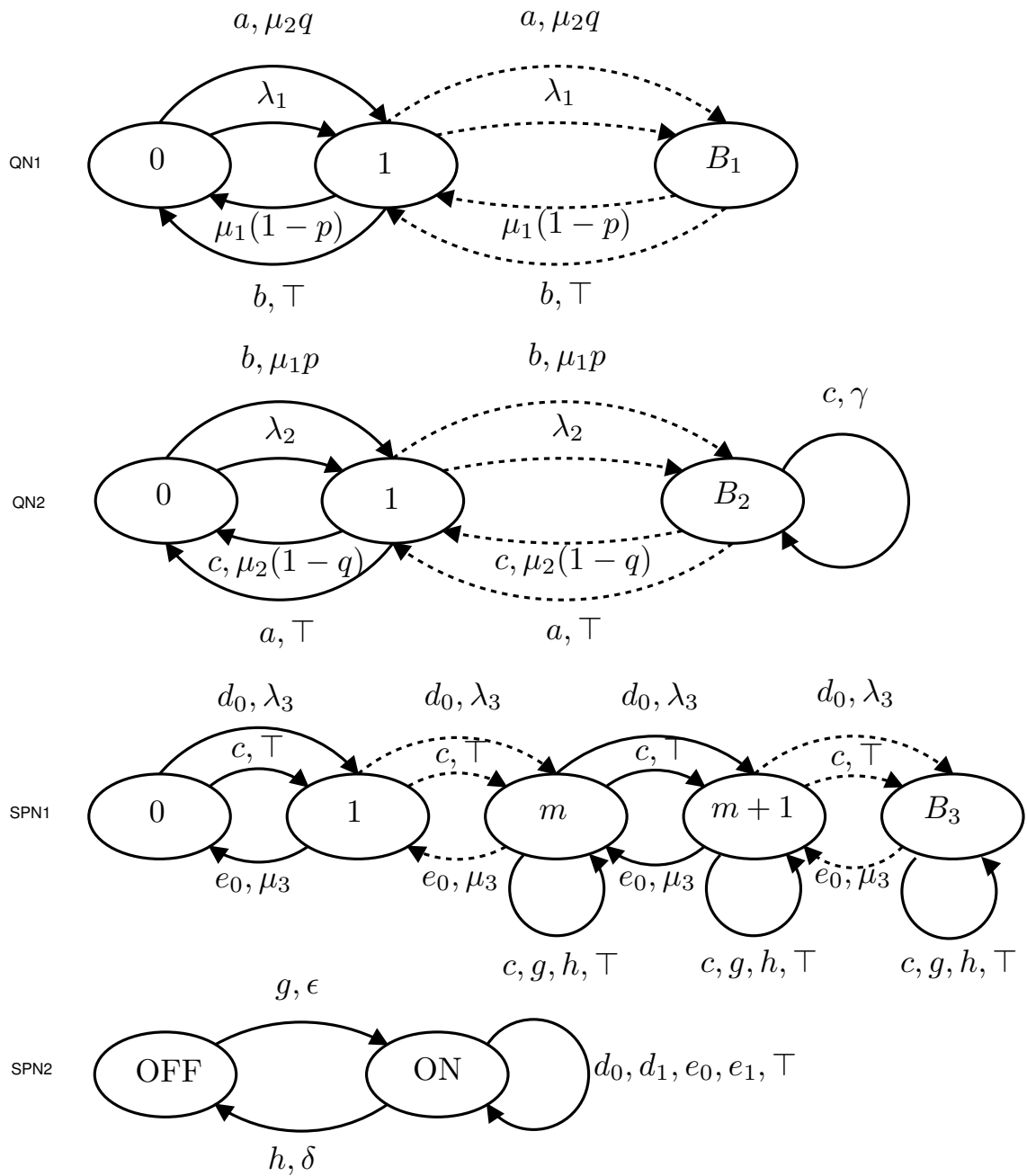


Figure 9.7: Labeled Exponential Automata produced by SIMTHESysER

INAP numerically solve the model in isolation and then checks Condition 9.1) for each ergodic joint state computed by SIMTHESys. The time complexity for the model solution in isolation is proportional to the cube of each queue capacity, therefore if  $B_1 = B_2 = B_3 = B$  we have  $\mathcal{O}(kB^3)$ , where  $k$  is the number of iterations required to converge withing a certain precision. In our experiments, we observed  $k \leq 9$  for all the parameterisation we tried. Due to the sparsity of the transitions in the joint state, Condition 9.1 can be checked in constant time for each joint-state, therefore leading to a total time complexity of  $\mathcal{O}(B^3)$ . We can conclude that in this case the product-form analysis reduces the time complexity of the solution from  $\mathcal{O}(B^9)$  to  $\mathcal{O}(B^3)$ .

## 9.4 Conclusions and future work

In this chapter we have presented a technique to identify product form solutions in a multiformalism framework. The check for the existence and the computation of the solution is performed during the generation of the state space of the submodels that composes the main model under study. State space is generated using behaviours that are associated with formalism specification. This provides a greater flexibility and allows the application of product form solutions to new formalism without the development of new tools.

Future work will include the study of more complex synchronization mechanisms, and the use of the product forms solutions to approximate real solution when the necessary conditions are not respected.



---

# 10

## Modelling retrieval-upon-conflict systems with product-form stochastic Petri nets

### 10.1 Introduction

In section 2.2 we have recalled some notions about *stochastic Petri nets*, and we have noted that their underlying stochastic process is a Continuous Time Markov Chain (CTMC).

However, it can be shown that the cardinality of the model's state space can grow more than exponentially with the structure of the model, i.e., the number of places and transitions and hence the whole generation is time and space consuming. Even worse, the numerical algorithms for deriving the steady-state performance indices become numerically unstable and prohibitive in terms of computation time. In order to overcome this problem, known as *state space explosion*, a range of techniques has been proposed, from state space reduction through aggregation (lumping) to approximation techniques.

As we already considered for other formalisms, product-form theory takes another approach: applying a *divide et impera* paradigm, SPNs can be efficiently solved through the analysis of their components (i.e., the places) in isolation. Product-form SPNs were first introduced in [126], and then generalised in [39, 95, 56]. More recently, new theoretical developments have been proposed connected to system biology [130]. We base our chapter on the methodology and results developed in [17, 131], in which a general approach to design and recognise product-form stochastic Petri nets, based on the Reversed Compound Agent Theorem (RCAT) [89] and its extensions [91, 92], is described. The analysis of the class of SPNs proposed in [95, 56] relies on a so called *rank theorem* that imposes a strict condition of the transition rates which is quite hard to interpret from the modelling point of view. In [17, 131] the authors show that that class of product-form SPNs (extended to open nets) can be analysed in terms of decomposition into simpler structures and hence the rate condition can be interpreted locally rather than on the whole SPN.

The aim of this chapter, based on a work we published in [14], is to analyse

a class of SPNs that is useful to model systems in which concurrent activities can lead to conflicts, requiring a recovery phase before a new execution of the same activities is retried. Instances of this kind of systems are quite frequent in the real world, for example in computer networks, databases and operating systems. We provide a formal model for these systems in terms of SPNs and show that they belong to the class studied in [17, 131]. Moreover, we prove two interesting properties for such a class of SPNs: first, their product-form does not require any condition on the transition rates and, second, the joint state space is the Cartesian product of the states that are reachable by each of the model's places. The former property enhances the applicability of the proposed model, while the latter allows us to derive the normalised stationary distribution in a straightforward way for open models. The model that we proposed can be combined with other quasi-reversible components maintaining the product-form property of the joint steady-state distribution (see, e.g., [26, 128, 78]).

For the theoretical background needed to understand this chapter, we refer to Section 2.2.

## 10.2 The conflict model

Consider a SPN consisting of a set of interconnecting building blocks: a *main building block* (MBB) with  $l$  places (from here onward MBB( $l$ )) and a certain number of *conflict building blocks* (CBB).

As previously stated, the building block MBB( $l$ ) has a set  $L$  of  $l$  places  $L = \{P_1, \dots, P_l\}$ . For each place  $P_i$ , there are both an incoming transition  $T_i$  with rate  $\lambda_{P_i}$  and an outgoing transition  $T'_i$  with rate  $\mu_{P_i}$ . Examples of the main building block with 2 or 3 places, respectively, are given in Figures 10.1 and 10.2. For each subset  $C$  of two or more places,  $C \subseteq L$ ,  $|C| \geq 2$ , we define an incoming transition  $T_C$  with rate  $\lambda_C$  and an outgoing transition  $T'_C$  with rate  $\mu_C$ . For each of those pairs of transitions, there is a conflict building block defined as follows: an arc connects transition  $T'_C$  to the place  $P_C$ , which is in turn connected to transition  $T_C$ . Notice that the firing semantics of transitions  $T_C$ , with  $|C| \geq 2$ , can be single server or infinite servers [134]. A conflict building block in isolation is shown in Figure 10.3, while a complete model for  $l = 2$  is depicted in Figure 10.4.

If  $|C| = k$ , there are  $\binom{l}{k}$  of such CBBs, and thus the total number of CBBs is  $\sum_{k=2}^l \binom{l}{k} = 2^l - l - 1$ . The total number of places in the model is then  $|\mathcal{P}| = 2^l - 1$ , and the total number of transitions is  $|\mathcal{T}| = 2|\mathcal{P}| = 2^{l+1} - 2$ .

**Example 12** (Conflict model for 2 classes). *Let us consider the SPN model depicted by Figure 10.4. Customers arrive at  $P_1$  and  $P_2$  according to independent homogeneous Poisson processes modelled by  $T_1$  and  $T_2$ , respectively. Successful customers' services are modelled by transitions  $T'_1$  and  $T'_2$ . Transition  $T'_{1,2}$  models the conflict events, and is enabled only when both  $P_1$  and  $P_2$  are non-empty. After a*

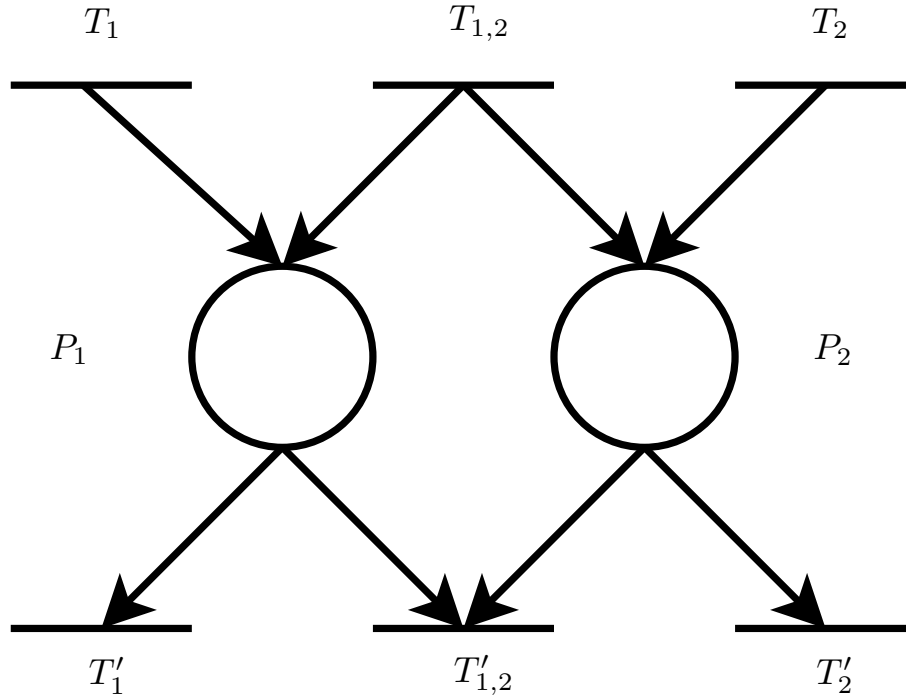


Figure 10.1: The main building block with 2 places

*conflict event a token is removed from each of the places  $P_1$  and  $P_2$  and one is put into  $P_{1,2}$  to represent the recovery phase. Transition  $T_{1,2}$  models the recovery time. This model is in product-form because the arrivals modelled by transitions  $T_1$  and  $T_2$  can be replaced by the output process of other product-form models (such as BCMP queues [26], G-queues [78] or MSCCC [128]) and, conversely, the firing of transitions  $T'_1$  and  $T'_2$  can be used to model the arrival process at other product-form models.*

The following proposition shows that conflict model is in product-form according to Theorem 1.

**Proposition 4** (Product-form of the conflict model). *The conflict model consists of building blocks satisfying the structural conditions of Theorem 1. Moreover, in stability, it yields without any rate-constraint the following product-form solution:*

$$\pi(\mathbf{m}) = \prod_{C \in 2^L \setminus \emptyset} g_C(m_C)$$

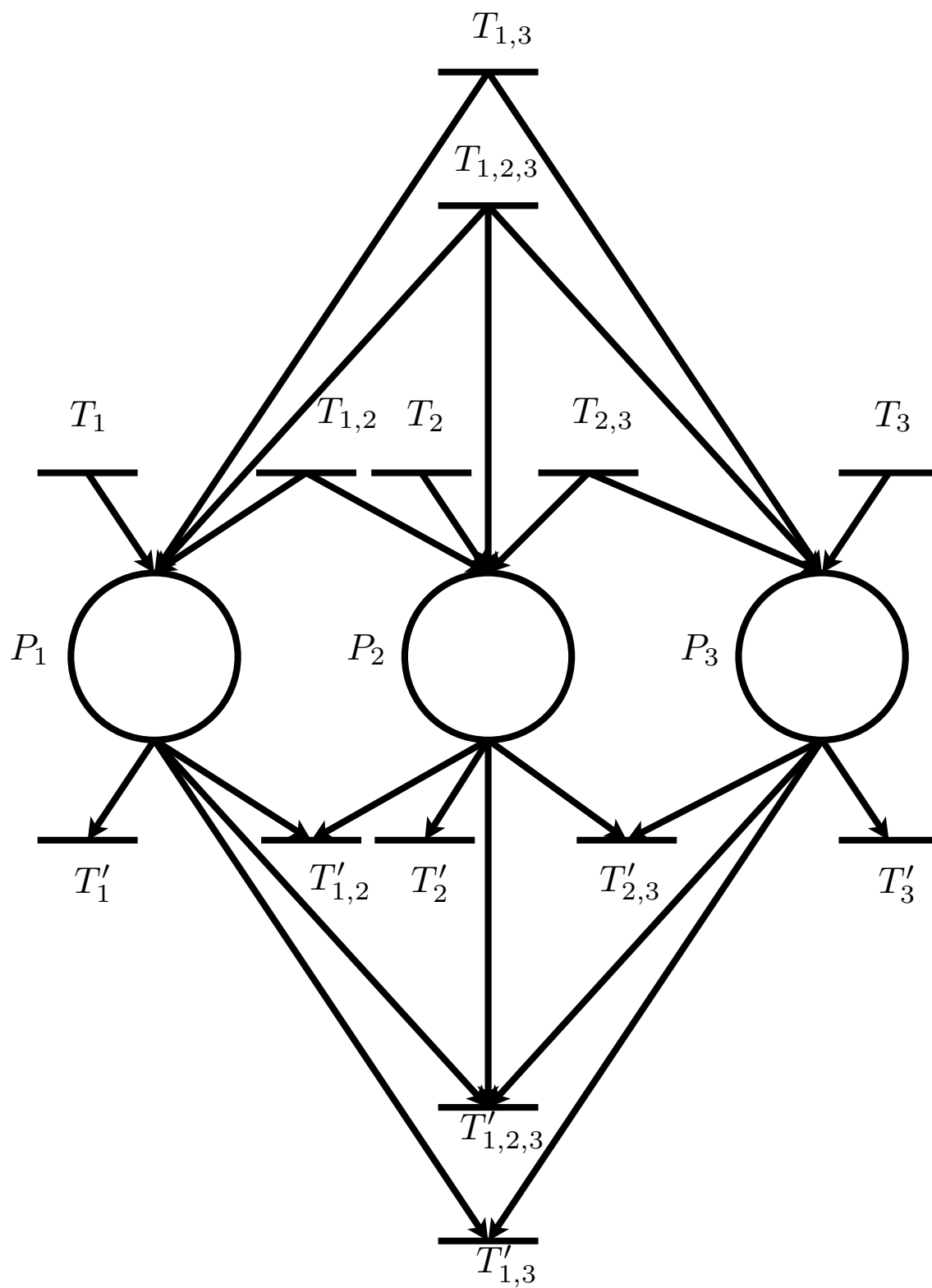


Figure 10.2: The main building block with 3 places



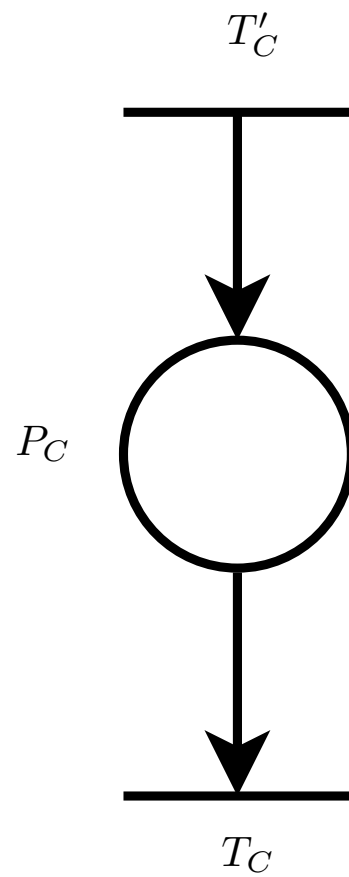


Figure 10.3: The conflict building block

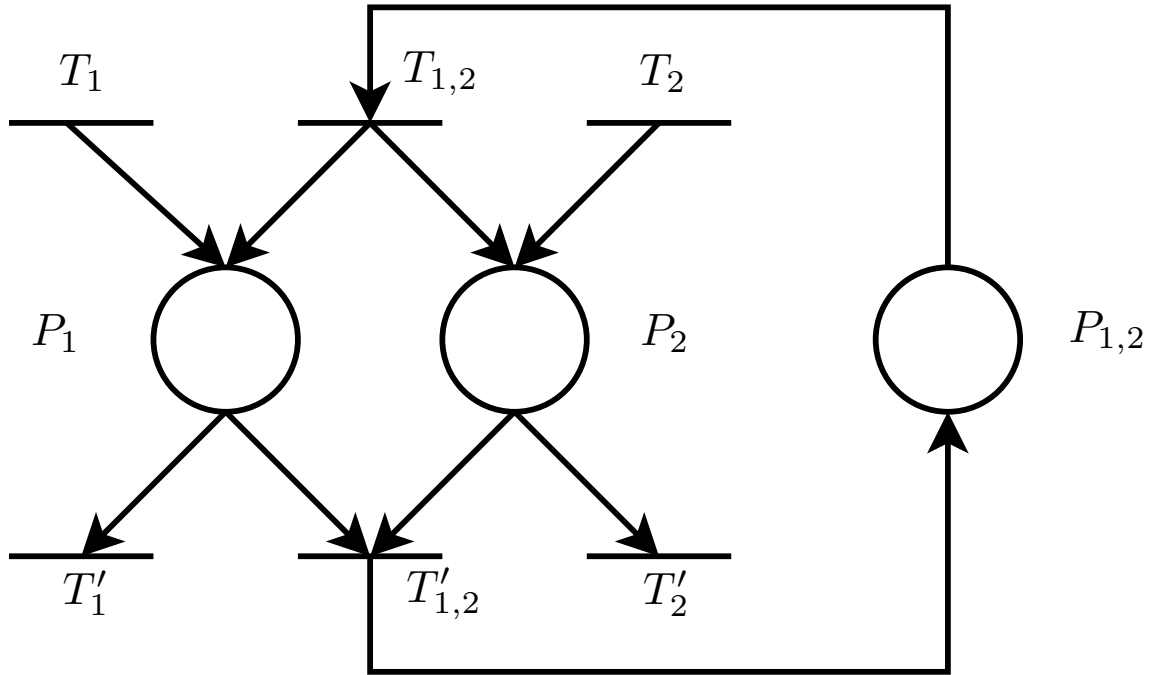


Figure 10.4: A complete model for  $l = 2$

where  $m_C$  is the component of the joint state associated with place  $P_C$  and

$$g_C(m_C) = \begin{cases} (1 - \frac{\lambda_P}{\mu_P})(\frac{\lambda_P}{\mu_P})^{m_P} & \text{if } C = \{P\} \\ (1 - \frac{\mu_C}{\lambda_C} \prod_{P \in C} \frac{\lambda_P}{\mu_P})(\frac{\mu_C}{\lambda_C} \prod_{P \in C} \frac{\lambda_P}{\mu_P})^{m_C} & \text{if } |C| \geq 2 \text{ and } T_C \text{ is single server} \\ (\frac{\mu_C}{\lambda_C} \prod_{P \in C} \frac{\lambda_P}{\mu_P})^{m_C} \exp(-\frac{\mu_C}{\lambda_C} \prod_{P \in C} \frac{\lambda_P}{\mu_P}) \frac{1}{m_C!} & \text{if } |C| \geq 2 \text{ and } T_C \text{ is } \top \text{ servers} \end{cases}$$

*Proof.* First, we prove that the rate conditions of Theorem 1 are always satisfied. Basically, we must show that:

$$\rho_C = \prod_{i \in C} \rho_i,$$

for any choice of  $\lambda_C$  and  $\mu_C$ , with  $C \in 2^L \setminus \emptyset$ . Indeed, the building blocks  $P_C$  behave like a  $M/M/1$  or  $M/M/\top$  queues and hence it is well-known that they are unconditionally in product-form. Observe that, transition  $T_C$  fires only when  $P_C$  has at least one token, therefore its throughput is *not*  $\lambda_C$  but its reversed rate  $\bar{\lambda}_C$  (according to RCAT terminology). Therefore, in order to satisfy Conditions (2.1) we must have  $\bar{\lambda}_C = \mu_C \prod_{i \in C} \lambda_i / \mu_i$ . Let us now consider the rate equation corresponding to the reversed rate (throughput) of transition  $T'_C$ . Using Lemma 1, we immediately derive that its throughput, i.e., the arrival rate perceived by  $P_C$ , is  $\bar{\lambda}_C$ . Observe that, analogously to what happens in closed queueing networks, the flows of tokens due to the firing of  $T_C$  and  $T'_C$  with  $C \geq 2$  give the identity  $\bar{\lambda}_C = \bar{\lambda}_C$  and hence

any non-trivial solution can be taken and, here we take:  $\bar{\lambda}_C = \prod_{i \in C} (\lambda_i / \mu_i) \mu_C$  [17]. However, differently from closed queueing networks, the conflict model's state space is such that each of its places can have from 0 to  $\top$  tokens, and the joint state space is the Cartesian product of the state spaces of each of its components. Therefore, the normalisation can be carried out by normalising the stationary distribution of each place considered in isolation.  $\square$

For the stability conditions, the following proposition holds:

**Proposition 5.** *The conflict model is stable if the following conditions hold:*

$$\forall i \in \{1, \dots, l\} \quad \lambda_i < \mu_i, \quad (10.1)$$

for the places of the main building block, while for the places of conflict building blocks  $P_C$  whose corresponding  $T_C$  is single server, we have that

$$\forall C \subseteq L \quad \bar{\mu}_C = \mu_C \prod_{P_i \in C} \rho_{P_i} < \lambda_C, \quad (10.2)$$

where  $\bar{\mu}_C$  identifies the throughput (reversed rate) of transition  $T_C$ .

*Proof.* The proof relies on deriving the necessary and sufficient conditions required for normalising the stationary probabilities of Proposition 4.  $\square$

Notice that Condition (10.2) can be rewritten as

$$\forall C \subseteq L \quad \mu_C \frac{\prod_{P_i \in C} \lambda_{P_i}}{\prod_{P_i \in C} \mu_{P_i}} < \lambda_C \quad (10.3)$$

If all the transitions  $T_C$  have a single server semantics, we can compute the average number of customers in the system as

$$E[N] = \sum_{i=1}^l \frac{\rho_{P_i}}{1 - \rho_{P_i}} + \sum_{C \subseteq L, |C| \geq 2} |C| \frac{\rho_C}{1 - \rho_C} \quad (10.4)$$

Now we consider the case where all the transitions  $T_C$ , with  $|C| \geq 2$  have an *infinite server* semantics, i.e. they can be seen as delay stations [134]. In this case the average number of customers for each place  $P_C$  becomes  $\rho_C$ , and thus the average number of customer for the whole system is

$$E[N] = \sum_{i=1}^l \frac{\rho_{P_i}}{1 - \rho_{P_i}} + \sum_{C \subseteq L, |C| \geq 2} |C| \rho_C \quad (10.5)$$

In both cases, the average response time of the system is given by Little's law, i.e., under their respective stability conditions:

$$E[R] = \frac{E[N]}{\sum_{i=1}^l \lambda_i} \quad (10.6)$$

## 10.3 Applications

In this section we present two applications of the class of stochastic Petri nets that we just described. Notice that, in order to model those systems, we have to make some assumptions in their probabilistic behaviour. An analysis on the robustness of assumptions with respect to the performance indices can be found in [127].

### 10.3.1 A computer network with collisions

We want to model a computer network in which there is a set  $L$  of  $l$  transmitting stations  $L = \{s_1, \dots, s_l\}$ . Packets become ready to be sent from each station  $s_i$  according to an homogeneous Poisson distribution with parameter  $\lambda_i$ . The global rate at which packets arrive to the system is thus  $\lambda = \sum_{i=1}^l \lambda_i$ . The time that a packet takes to be transmitted from each station  $s_i$  is exponentially distributed with parameter  $\mu_i^*$ . The channel, by itself, is capable of transmitting with a global rate of  $M = \sum_{i=1}^l \mu_i^*$ . During the transmission of a packet, a collision, i.e., the simultaneous transmission of more than one packet using the same physical medium, can occur, thus making the transmission ineffective. A collision can occur between any combination of  $k$  stations,  $2 \leq k \leq L$ , with probability  $p_k(L)$ . After a collision between the transmissions of a subset of stations  $C \subseteq L$ ,  $|C| \geq 2$ , an exponentially-distributed recovery time, with parameter  $\mu_C$  is performed. After that time, a new transmission is retried.

We can abstract the previously described system as a *conflict model* (Section 10.2), in which, since there is no queueing for recovery phases, the conflict building blocks have an infinite-server firing semantics.

If we assume that  $\mu_{s_i} = \mu_1$ ,  $\lambda_{s_i} = \lambda_1$ ,  $\forall s_i \in L$  and that  $\lambda_C = \lambda_{|C|}$  and  $\mu_C = \mu_{|C|}$ ,  $\forall C \subseteq L$ ,  $|C| \geq 2$ , i.e., all the transitions representing collisions between the same number of stations have the same rate, we can simplify Equation (10.4) as

$$E[N] = l \frac{\rho_1}{1 - \rho_1} + \sum_{k=2}^l \binom{l}{k} k \rho_k \quad (10.7)$$

We now have to make further assumptions on the behaviour of the system in order to parametrise the model and thus compute the numerical value of the interested performance indices.

Let  $q \in [0, 1]$  be the probability that a certain station is transmitting on the channel. Here we assume that  $q$  can be computed as:

$$q = \frac{\lambda_1}{M}, \quad (10.8)$$

such that  $q^2$  represents the probability of a collision between two nodes. We define  $p_C(L)$ , i.e., the probability of having a collision between a specific set  $C$  of  $k$  stations in a system with  $L$  stations, as

$$p_C(L) = q^k (1 - q)^{L-k} \quad (10.9)$$

Parameter Name	Value
$M$	$10^8/(800 \cdot 8) \approx 15625$
$\lambda_C$	$10^8/512 \approx 195313$

Table 10.1: Parameter values for the example of Section 10.3.1

and, since, for any given station, there are  $\binom{L-1}{k-1}$  possible collision sets of  $k$  station, the probability  $p_k(L)$  of having a collision among  $k$ , having chosen one of the station, is

$$p_k(L) = \binom{L-1}{k-1} q^k (1-q)^{L-k} \quad (10.10)$$

Thus, the probability  $\bar{p}$  of not having any collision for a chosen station is

$$\bar{p} = 1 - \sum_{k=2}^L \binom{L-1}{k-1} q^k (1-q)^{L-k} \quad (10.11)$$

We can now compute the various service rates  $\mu_C$ ,  $C \subset L$ ,  $|C| = k \geq 2$  as

$$\mu_k = \mu^* q^k (1-q)^{L-k} \quad (10.12)$$

and for  $\mu_1$

$$\mu_1 = \mu^* \left( 1 - \sum_{k=2}^L \binom{L-1}{k-1} q^k (1-q)^{L-k} \right) \quad (10.13)$$

### A numerical example

Consider an Ethernet-like network having a shared channel with a bandwidth of 100Mbps. We assume that the time to send a frame is exponentially distributed, with an expected value that is the time to send 800 bytes at full speed, and that, on average, the backoff time is the same as the one required for transmitting 512 bits. We can model the network as described in Section 10.3.1, having a set of parameters given in Table 10.1,

Fixed a value for  $L$ , i.e., the number of transmitting stations, we set the other parameters according to Equations (10.8), (10.12) and (10.13), while  $\mu_1^* = ML^{-1}$

In Figure 10.5 we see the behaviour of the average response time, i.e., the average time a packet has to spend to be successfully transmitted, as the value of  $\lambda$  increases, with three different numbers of stations  $L$ . We can see that, the response time increases both as the arrival rate increases, due to queueing, and as the number of stations increases, due to collisions.

In Figure 10.6 we can see how, given a fixed arrival rate at each station,  $\lambda_1 = 300$ , the average response time increases with the number of transmitting stations, due to the increased number of collisions.

12010. Modelling retrial-upon-conflict systems with product-form stochastic Petri nets

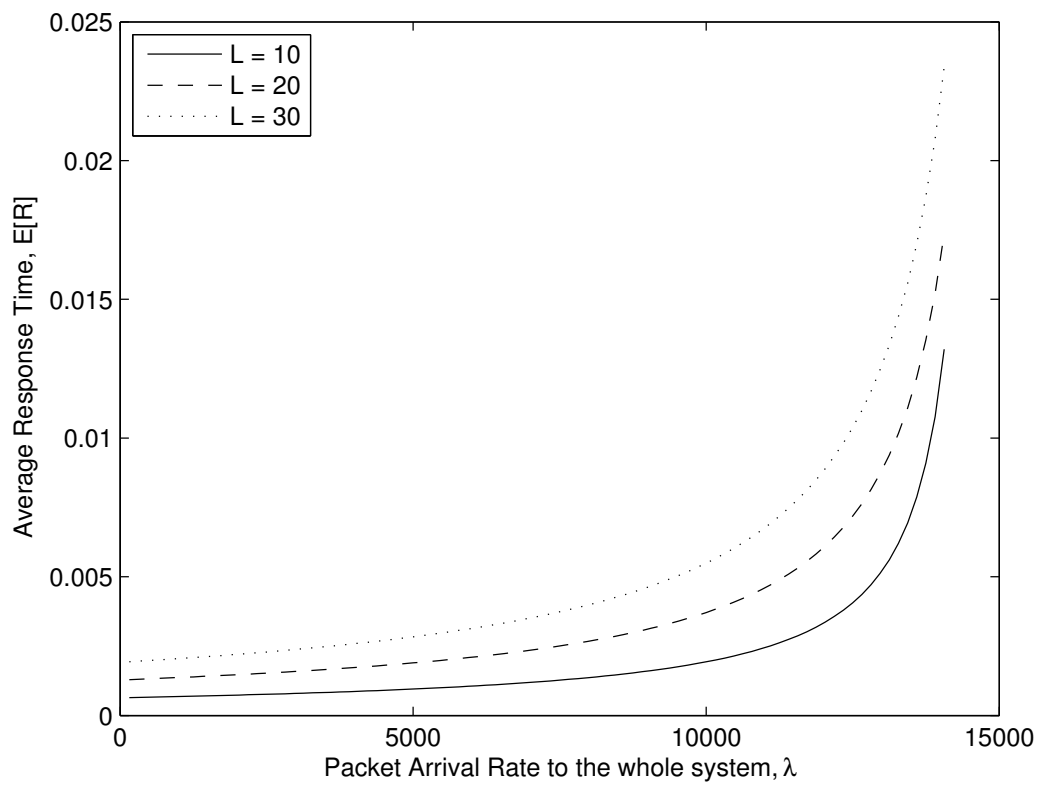


Figure 10.5: Average response time as a function of packet arrival rate, with different number of stations

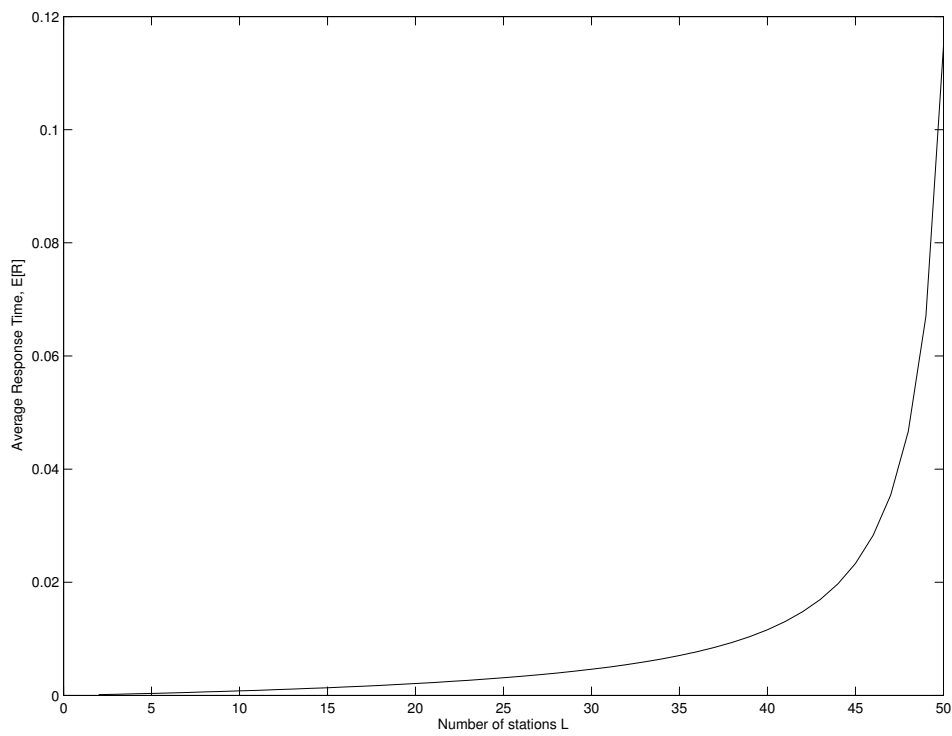


Figure 10.6: Average response time as a function of the number of stations

### 10.3.2 A transactional database system

Consider a database system in which there is a set  $L$  of  $l$  processors  $L = \{s_1, \dots, s_l\}$ . The transaction requests to be processed arrive to the processor  $s_i$  according to an homogeneous Poisson distribution with parameter  $\lambda_i$ . The time that a transaction takes to be processed by  $s_i$  is exponentially distributed with parameter  $\mu_i^*$ . During the execution of a transaction, a conflict, i.e., a concurrent access to the same data in which at least one of the access is a write operation, can occur, thus requiring a strategy to resolve the conflict. A collision can occur between any combination of  $k$  stations,  $2 \leq k \leq L$ , with probability  $p_k(L)$ . In order to implement this strategy, after a conflict occurs between the transactions running on a subset of processors  $C \subseteq L$ ,  $|C| \geq 2$ , an exponentially-distributed recovery time, with parameter  $\mu_C$  is performed. If the same set of processors is engaged in the resolution of another conflict, the resolution requests are enqueued. After the conflict is recovered, the transaction is retried.

As for the example of Section 10.3.1, we can model the described system as a conflict model. However, in this case, since recovery requests are enqueued, the conflict building blocks have the ordinary firing semantics of SPNs.

As for the previous example, we can assume that  $\mu_{s_i} = \mu_1$ ,  $\lambda_{s_i} = \lambda_1$ ,  $\forall s_i \in L$  and that  $\lambda_C = \lambda_{|C|}$  and  $\mu_C = \mu_{|C|}$ ,  $\forall C \subseteq L, |C| \geq 2$ , i.e., transitions of conflict building blocks representing recoveries between the same number of processors have the same rate. Thus we can simplify Equation (10.4) as

$$E[N] = \sum_{k=1}^l \binom{L}{k} k \frac{\rho_k}{1 - \rho_k} \quad (10.14)$$

Moreover, we observe that the same arguments made for the collision probabilities could be made for transaction conflicts, assuming that the collision probabilities, here, mean the probability of having a conflicting transaction between  $k$  processors. Thus, Equations 10.12 and 10.13 still hold. However, here the choice of a suitable parameter  $q$  is free.

#### A numerical example

Consider a database system like the one described in Section 10.3.2, in which each processor is capable of serving, on average, 100 transactions per second and the system can recover 10 transactions per second. The parameters are shown in Table 10.2, where  $q$  is chosen arbitrarily.

Figure 10.7 shows the average response time as a functions of the arrival rate of transactions to each processor.

Moreover, in systems where  $q$  is constant, from Equations (10.1) and (10.13) we directly obtain the maximum admissible arrival rate to each processor  $\lambda_1$  as a function of  $q$ ,  $\mu$  and  $L$ . Figure 10.8 shows a numerical example for  $\mu_1 = 1$ .



Parameter Name	Value
$\mu_1$	100
$\lambda_C$	10
$q$	$10^{-2}$

Table 10.2: Parameter values for the example of Section 10.3.2

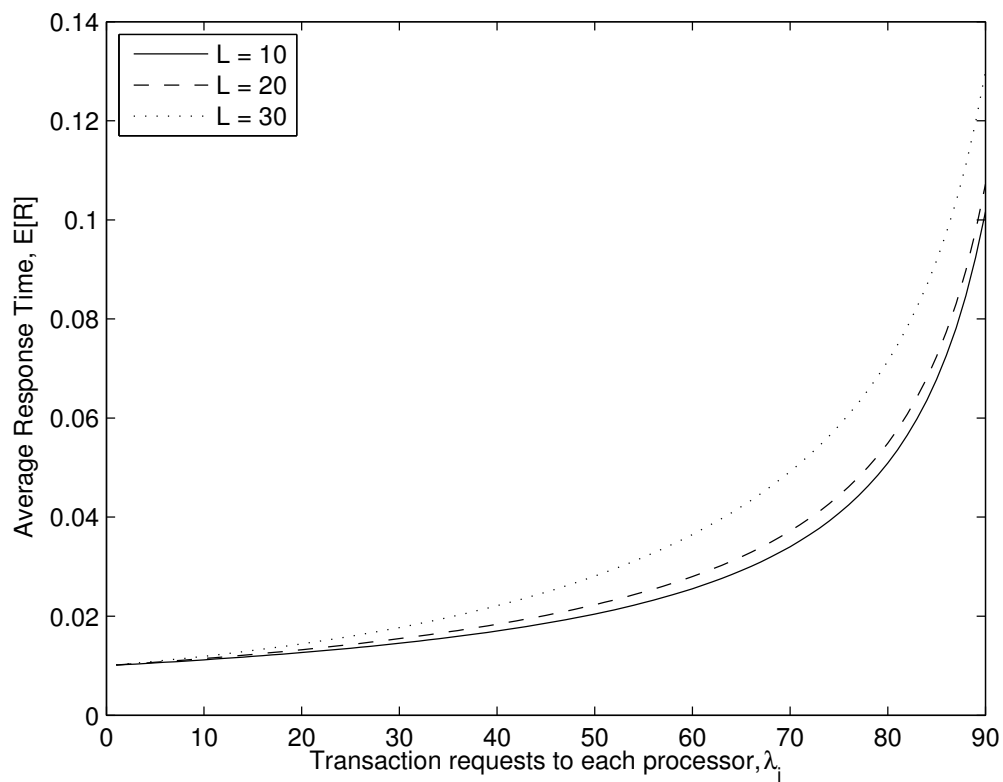


Figure 10.7: Average response time as a function of the arrival rate of transactions, with different number of processors

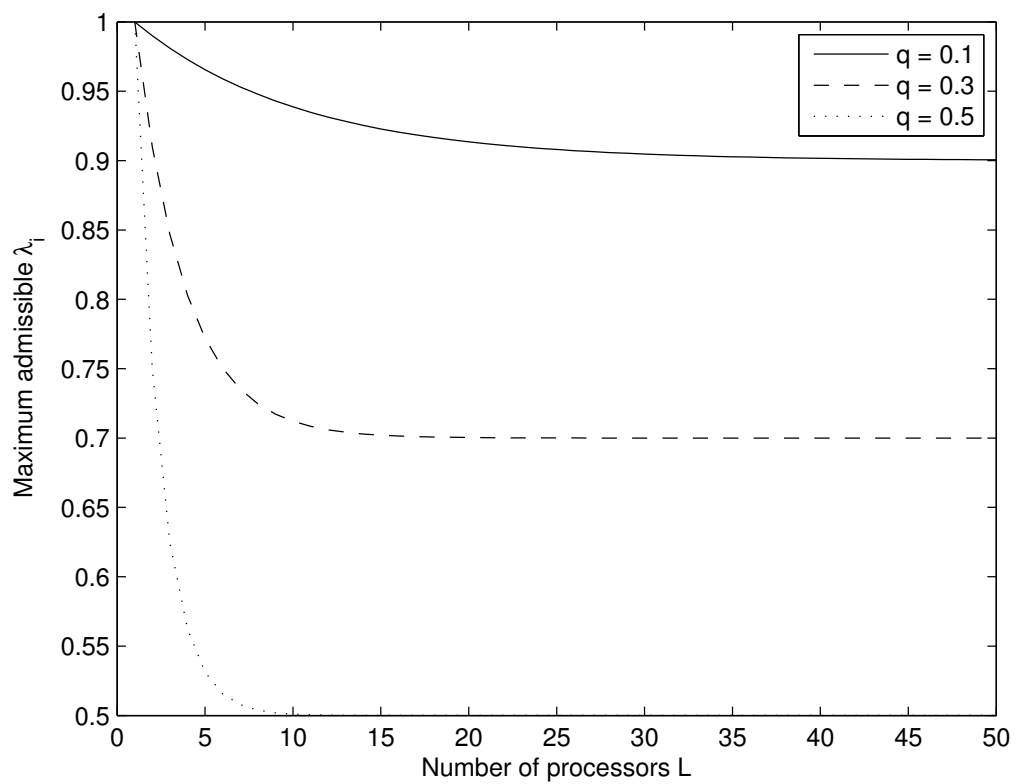


Figure 10.8: Maximum arrival rate to each processor as a function of the number of processors, with different conflict probabilities

## 10.4 Conclusions

In this chapter we have proposed a conflict model for the analysis of systems with retrial-upon-conflict strategies. The salient features of the conflict models are:

- It is formally defined in terms of stochastic Petri nets.
- The stationary distribution can be computed efficiently because the conflict model is product-form. Moreover, it can be composed with other models that are in product-form by RCAT or that have been proved to be quasi-reversible, including the BCMP stations [26], the G-queues [78] and other more sophisticated queueing stations [128].
- We have shown that, differently from other SPNs, the conflict model is unconditionally in product-form. In many other cases, when dealing with product-form SPNs, rate conditions arise (see [95, 56, 130, 17, 131]).

We have shown the application of the conflict model to study some realistic systems and have given some numerical results which show examples on how those models behave under some conditions.

**Future Works** We plan to extend our work in order to consider also closed networks with conflict models, i.e., systems where the total number of customers remain constant. In these cases the problem of the computation of the normalising constant must be addressed. The well-known algorithms for the direct computation of the normalising constant (e.g., the convolution [48, 56]) or for the computation of the average performance indices [151, 160] should be considered. Moreover, the problem of parametrising the model according to the behaviour of the system, especially due to the combinatorial nature of these model, should be studied in depth.

**12610. Modelling retrial-upon-conflict systems with product-form stochastic Petri nets**

---

# Conclusions

In this thesis we presented some of the results that was obtained during my 3 years as a Ph.D. student. In particular, we focused only on the works that were related to the efficient solution of stochastic models for which a *cooperation* semantics is defined.

The publications whose content was partially included in this thesis were [15, 9, 12, 11, 7, 13, 22, 14].

As stated in the Preface, some other publications were not included in the thesis, since they were not directly related to the solution methods of cooperating stochastic models, namely [8, 10, 66, 67, 77, 65].

## C.1 Contributions

After an introduction to the background notions needed to understand the thesis (Part I), we moved to consider the main theoretical and practical contributions of the thesis (Part II), which are in distinct, but related, fields.

Chapter 5 deals with automated product form detection and solution of models described as a set of cooperating components, which can possibly have an infinite state space. In this context we proposed an algorithm able to perform both tasks, and a tool, equipped with a graphical user interface, that allows users to design and analyse such models.

Chapter 6 deals with a different approach to the efficient solution of cooperating stochastic models, i.e., state space aggregation (lumping). We proposed a set of sufficient conditions under which such aggregation can be done component-wise such that the composition of aggregated models is, indeed, an aggregation of the composition of the original ones.

Chapter 7 puts, through the introduction of the concept of *conditional product form*, the content of Chapters 5 and 6 in relation. We proved that finding a reversible aggregation of the reversed process of one of a pair of components simplifies the computation of the steady state probability distribution of the joint model.

Finally, in Chapter 8 we presented some approximation techniques for models that cannot be exactly aggregated according to the results of the previous chapters.

## C.2 Impact and Future Works

The theoretical results and the techniques presented in Part II can be applied to a variety of problems. In Part III we presented some applicative results obtained

using the aforementioned methods, in particular regarding models expressed using multiple formalisms (Chapter 9) and stochastic Petri Nets (Chapter 10).

As concern future works, we are working on multiple fields, such as on general convergence proofs for the algorithm of Chapter 5, and on practical approximate aggregation algorithm for the method proposed in Chapter 8. A current work is ongoing on finding practical examples of models exhibiting a *conditional product-form* (Chapter 7).

---

# Bibliography

- [1] I. F. Akyildiz. Exact product form solution for queueing networks with blocking. *IEEE Trans. on Comp.*, C-36-1:122–125, 1987.
- [2] S. Baarir, M. Beccuti, C. Dutheillet, G. Franceschinis, and S. Haddad. Lumping partially symmetrical stochastic models. *Perf. Eval.*, 68(1):21 – 44, 2011.
- [3] F. Baccelli, W.A. Massey, and D. Towsley. Acyclic fork-join queueing networks. *J. ACM*, 36(3):615–642, 1989.
- [4] Jos CM Baeten and W Peter Weijland. *Process algebra, volume 18 of Cambridge tracts in theoretical computer science*. Cambridge University Press, 1990.
- [5] G. Balbo, S. C. Bruell, and M. Sereno. On the relations between BCMP Queueing Networks and Product Form Solution Stochastic Petri Nets. *Proc. of 10th Int. Workshop on Petri Nets and Performance Models, 2003.*, pages 103–112, 2003.
- [6] S. Balsamo, V. De Nitto Persone', and R. Onvural. *Analysis of Queueing Networks with Blocking*. Kluwer Academic Publishers, 2001.
- [7] S. Balsamo, G. Dei Rossi, and A. Marin. Efficient solutions for cooperating automata based on forward and reversed lumping. Technical Report DAIS-2012-6, Dip. di Scienze Ambientali, Informatica e Statistica, Università Ca' Foscari Venezia.
- [8] S. Balsamo, G. Dei Rossi, and A. Marin. Applying BCMP multi-class queueing networks for the performance evaluation of hierarchical and modular software systems. In *Proc. of Int. Conf. ESM 2010*, pages 206–213, Hasselt, BE, 2010. Eurosis.
- [9] S. Balsamo, G. Dei Rossi, and A. Marin. A numerical algorithm for the solution of product-form models with infinite state spaces. In *Proc. of EPEW 2010: Comp. Perf. Eng.*, pages 191–206, Bertinoro, Italy, 2010. LNCS 6342/2010.
- [10] S. Balsamo, G. Dei Rossi, and A. Marin. Optimisation of virtual machine garbage collection policies. In *LNCS 6751/2011, proc. Int. Conf. ASMTA*, pages 70–84, Venice, IT, 2011. Springer.
- [11] S. Balsamo, G. Dei Rossi, and A. Marin. Cooperating stochastic automata: approximate lumping an reversed process. In *Comp. and Inf. Sciences III, ISCIS*, pages 131–141, Paris, FR, 2012. Springer.

- [12] S. Balsamo, G. Dei Rossi, and A. Marin. Lumping and reversed processes in cooperating automata. In *LNCS 7314, Proc. of Int. Conf. ASMTA*, pages 212–226, Grenoble, FR, 2012. Springer.
- [13] S. Balsamo, G. Dei Rossi, and A. Marin. A survey on multi-formalism performance evaluation tools. In *Proc. of Int. Conf. ESM*, pages 15–23, Essen, DE, 2012. Eurosis.
- [14] S. Balsamo, G. Dei Rossi, and A. Marin. Modelling retrial-upon-conflict systems with product-form stochastic petri nets. In *LNCS 7984, Proc. of Int. Conf. ASMTA*, pages 52–66, Gent, BE, 2013. Springer.
- [15] S. Balsamo, G. Dei Rossi, and A. Marin. A tool for the numerical solution of cooperating Markov chains in product-form. In *Proc. Of Int. Conf. HET-NETs*, pages 311–324, Zakopane, PL, January, 2010.
- [16] S. Balsamo, P. G. Harrison, and A. Marin. A unifying approach to product-forms in networks with finite capacity constraints. In Vishal Misra, Paul Barford, and Mark S. Squillante, editors, *Proc. of the 2010 ACM SIGMETRICS Int. Conf. on Measurement and Modeling of Computer Systems*, pages 25–36, New York, NY, USA, 14-18 June 2010.
- [17] S. Balsamo, P. G. Harrison, and A. Marin. Methodological construction of product-form stochastic Petri nets for performance evaluation. *Journal of Systems and Software*, 85(7):1520–1539, 2012.
- [18] S. Balsamo and G. Iazeolla. An extension of Norton’s theorem for queueing networks. *IEEE Trans. on Software Eng.*, SE-8:298–305, 1982.
- [19] S. Balsamo, A. Di Marco, P. Inverardi, and M. Simeoni. Model-based performance prediction in software development: a survey. *IEEE Trans. on Software Eng.*, 30(5):295–310, May 2004.
- [20] S. Balsamo and A. Marin. *Queueing Networks in Formal methods for performance evaluation*, chapter 2, pages 34–82. M. Bernardo and J. Hillston (Eds), LNCS, Springer, 2007.
- [21] Simonetta Balsamo and Andrea Marin. Performance engineering with product-form models: efficient solutions and applications. In *Proc. of the second joint WOSP/SIPEW international conference on Performance Eng.*, ICPE ’11, pages 437–448, New York, NY, USA, 2011. ACM.
- [22] E. Barbierato, G. Dei Rossi, M. Gribaudo, M. Iacono, and A. Marin. Exploiting product form solution techniques in multiformalism modeling. In *ENTCS, Proc. of Int. Workshop PASM*, page to appear, London, UK, 2012. Elsevier.



- [23] E. Barbierato, M. Gribaudo, and M. Iacono. Exploiting multiformalism models for testing and performance evaluation in SIMTHESys. In *Proceedings of the 5th International ICST Conference on Performance Evaluation Methodologies and Tools*, VALUETOOLS '11, pages 121–130. ICST), 2011.
- [24] E. Barbierato, M. Gribaudo, M. Iacono, and S. Marrone. Performability modeling of exceptions-aware systems in multiformalism tools. In *Proceedings of the 18th international conference on Analytical and stochastic modeling techniques and applications*, ASMTA'11, pages 257–272. Springer, 2011.
- [25] Enrico Barbierato, Marco Gribaudo, and Mauro Iacono. Defining formalisms for performance evaluation with SIMTHESys. *Electron. Notes Theor. Comput. Sci.*, 275:37–51, September 2011.
- [26] F. Baskett, K. M. Chandy, R. R. Muntz, and F. G. Palacios. Open, closed, and mixed networks of queues with different classes of customers. *J. ACM*, 22(2):248–260, 1975.
- [27] F. Bause. Queueing Petri nets: A formalism for the combined qualitative and quantitative analysis of systems. In *Proc. of 5th Int. Workshop on Petri Nets and Performance Models*, pages 14–23, Toulouse (France), 1993.
- [28] F. Bause, P. Kemper, and P. Kritzinger. Abstract Petri net notation. *Petri Net Newsletter*, (49):9–27, Oct 1995.
- [29] Falko Bause, Peter Buchholz, and Peter Kemper. A toolbox for functional and quantitative analysis of ded.s. In *Computer Performance Evaluation*, volume 1469 of *LNCS*, pages 356–359. Springer, 1998.
- [30] M. Bernardo, L. Donatiello, and R. Gorrieri. Modelling and Analyzing Concurrent Systems with MPA. In *Proc. of 2nd Process Algebra and Performance Modelling Workshop*, pages 175–189, 1994.
- [31] M. Bernardo, R. Gorrieri, and M. A. Zamboni. A tutorial on EMPA: A theory of concurrent processes with nondeterminism, priorities, probabilities and time. *Theoretical Computer Science*, 202:1–54, 1998.
- [32] M. Bernardo and J. Hillston. *Formal Methods for Performance Evaluation*, volume 4486. Springer, 2007.
- [33] Marco Bertoli, Giuliano Casale, and Giuseppe Serazzi. Jmt: performance engineering tools for system modeling. *SIGMETRICS Performance Evaluation Review*, 36(4):10–15, 2009.
- [34] DA Bini, B. Meini, S. Steffé, and B. Van Houdt. Structured Markov chains solver: software tools. In *Proc. of the 2006 workshop on tools for solving structured Markov chains*, page 14. ACM, 2006.

- [35] Henrik Bohnenkamp, Holger Hermanns, and Joost-Pieter Katoen. Motor: the modest tool environment. In *Proc. of the 13th international conference on Tools and algorithms for the construction and analysis of systems*, LNCS, pages 500–504. Springer, 2007.
- [36] P. Bonet, C. Llado, R. Puijaner, and W. Knottenbelt. Pipe v2.5.: a Petri net tool for performance modelling. In *Proc. of 23rd Latin American Conference on Informatics*, San Jose, Costa Rica, October, 2007.
- [37] Pere Bonet, Catalina M Lladó, Ramon Puijaner, and William J Knottenbelt. Pipe v2. 5: A petri net tool for performance modelling. In *Proc. 23rd Latin American Conference on Informatics (CLEI 2007)*, 2007.
- [38] R. Boucherie and N. M. van Dijk. Product-form queueing networks with state dependent multiple job transitions. *Advances in Applied Prob.*, 23:152–187, 1991.
- [39] R. J. Boucherie. A characterisation of independence for competing Markov chains with applications to stochastic Petri nets. *IEEE Trans. on Software Eng.*, 20(7):536–544, 1994.
- [40] J. T. Bradley and W. J. Knottenbelt. The ipc/HYDRA tool chain for the analysis of pepa models. In *Proc. of the 1st Int. Conf. on the Quantitative Evaluation of Systems (QEST)*, volume 4, pages 334–335, 2004.
- [41] J.T. Bradley, N.J. Dingle, S.T. Gilmore, and W.J. Knottenbelt. Derivation of passage-time densities in PEPA models using ipc: the imperial PEPA compiler. In *Modeling, Analysis and Simulation of Computer Telecommunications Systems, 2003. MASCOTS 2003. 11th IEEE/ACM International Symposium on*, pages 344–351. IEEE, 2003.
- [42] S. C. Bruell and G. Balbo. *Computational Algorithms for Closed Queueing Networks*. The Computer Science Library. Elsevier North Holland, 1980.
- [43] Giacomo Bucci, Luigi Sassoli, and Enrico Vicario. Correctness verification and performance analysis of real-time systems using stochastic preemptive time petri nets. *Tran. on Soft. Eng.*, 31(11):913–927, 2005.
- [44] P. Buchholz. Adaptive decomposition and approximation for the analysis of stochastic Petri nets. *Perf. Eval.*, 56:23–52, 2004.
- [45] P. Buchholz. Bounding stationary results of tandem networks with MAP input and MAP service time distributions. In *Proc. of ACM SIGMETRICS/PERFORMANCE*, pages 191–202, Saint Malo, FR, 2006.
- [46] P. Buchholz. Product form approximations for communicating Markov processes. *Perf. Eval.*, 67(9):797 – 815, 2010. Special Issue: QEST 2008.

- [47] P. Buchholz and P. Kemper. Numerical analysis techniques in the apnn toolbox. In *Workshop on Formal Methods in Performance Evaluation and Applications*, pages 1–6, 1999.
- [48] J. P. Buzen. Computational algorithms for closed queueing networks with exponential servers. *Commun. ACM*, 16(9):527–531, 1973.
- [49] K. M. Chandy, U. Herzog, and L. Woo. Parametric analysis of queueing networks. *IBM Journal of Res. and Dev.*, 1(1):36–42, 1975.
- [50] X. Chao. A queueing network model with catastrophes and product form solution. *Op. Res. Letters*, 18(2):75–79, 1995.
- [51] G. Chiola, G. Franceschinis, R. Gaeta, and M. Ribaud. Greatspn 1.7: Graphical editor and analyzer for timed and stochastic petri nets. *Perform. Eval., Elsevier*, 24:47–68, 1995.
- [52] G. Ciardo, R.L. Jones, AS Miner, and RI Siminiceanu. Logic and stochastic modeling with smart. *Performance Evaluation*, 63(6):578–608, 2006.
- [53] G. Ciardo and A. Miner. Storage alternatives for large structured state spaces. In *LNCS 1245, Computer Performance Evaluation Modelling Techniques and Tools*, pages 44–57. Springer, 1997.
- [54] G. Clark, T. Courtney, D. Daly, D. Deavours, S. Derisavi, J.M. Doyle, W.H. Sanders, and P. Webster. The mobius modeling tool. In *Proc. of the 9th Int. Workshop on Petri Nets and Performance Models*, pages 241–250. IEEE, 2001.
- [55] G. Clark and W.H. Sanders. Implementing a stochastic process algebra within the möbius modeling framework. In *Proc. of the Joint International Workshop on Process Algebra and Probabilistic Methods, Performance Modeling and Verification*, pages 200–216. Springer, 2001.
- [56] J. L. Coleman, W. Henderson, and P. G. Taylor. Product form equilibrium distributions and a convolution algorithm for Stochastic Petri nets. *Perf. Eval.*, 26(3):159–180, 1996.
- [57] Y. Colombani and S. Heipcke. Mosel: An extensible environment for modeling and programming solutions. In *Proc. of CP-AI-OR*, volume 2, pages 277–290, 2002.
- [58] A. E. Conway, E. de Souza e Silva, and S. S. Lavenberg. Mean Value Analysis by chain of product form queueing networks. *IEEE Trans. on Comp.*, 38(3):432–442, 1989.

- [59] A. E. Conway and N. D. Georganas. Recal - a new efficient algorithm for the exact analysis of multiple-chain closed queuing networks. *J. ACM*, 33(4):768–791, 1986.
- [60] A. E. Conway and N. D. Georganas. *Queueing Networks - Exact Computational Algorithms: A unified Theory Based on Decomposition and Aggregation*. The MIT Press, Cambridge, MA, 1989.
- [61] P.-J. Courtois and P Semal. Bounds for the positive eigenvectors of nonnegative matrices and for their approximations by decomposition. *J. of the ACM*, 31(4):804–825, 1984.
- [62] P.J. Courtois. *Decomposability*. Academic Press, New York, 1977.
- [63] Thu Ha Dao Thi and J. M. Fourneau. Stochastic automata networks with master/slave synchronization: Product form and tensor. In *Proceedings of the 16th International Conference on Analytical and Stochastic Modeling Techniques and Applications*, ASMTA '09, pages 279–293, Berlin, Heidelberg, 2009. Springer.
- [64] E. de Souza e Silva and S. S. Lavenberg. Calculating joint queue-length distributions in product-form queuing networks. *J. ACM*, 36(1):194–207, 1989.
- [65] G. Dei Rossi, L. Gallina, and S. Rossi. Performance analysis and formal verification of cognitive wireless networks. In *Proc. of EPEW 2013: Comp. Perf. Eng.*, page to appear. LNCS, Springer, 20103.
- [66] G. Dei Rossi and A. Marin. A queueing model with batch arrivals for studying the impact of fragmentation in wireless protocols. In *proc. Int. Conf. IFIP Wireless Days*, Niagara Falls, CA, 2011. IEEE.
- [67] G. Dei Rossi, A. Marin, M. Rosati, and S. Balsamo. A simulation package for an energy-aware comparison of arq protocols. In *Proc. of Int. Conf. ISC 2011*, pages 18–25, Venice, IT, 2011. Eurosis.
- [68] Nicholas J Dingle, William J Knottenbelt, and Tamas Suto. Pipe2: a tool for the performance evaluation of generalised stochastic petri nets. *ACM SIGMETRICS Performance Evaluation Review*, 36(4):34–39, 2009.
- [69] A. Economou. Generalized product-form stationary distributions for Markov chains in random environment with queueing application. *Adv. in Appl. Prob.*, 37:185–211, 2005.
- [70] Agner K. Erlang. The Theory of Probabilities and Telephone Conversations. *Nyt Tidsskrift for Matematik*, 20(B):33–39, 1909.

- [71] J. Esparza and M. Nielsen. Decidability Issues for Petri Nets - a Survey. *Bulletin of the European Association for Theoretical Computer Science*, 52:245–262, 1994.
- [72] J. M. Fourneau, B. Plateau, and W. J. Stewart. Product form for stochastic automata networks. In *Proc. of ValueTools '07*, pages 1–10, Brussels, Belgium, 2007. ICST.
- [73] J.M. Fourneau, L. Kloul, and F. Quessette. Multiclass G-networks with jumps back to zero. In *Proc. of MASCOTS '95*, pages 28–32, Durham, NC, USA, March 1995.
- [74] J.M. Fourneau and F. Quessette. Computing the steady-state distribution of G-networks with synchronized partial flushing. In *Proc. of ISCIS, 21th International Symposium*, pages 887–896, Istanbul, Turkey, 2006.
- [75] G. Franceschinis and Richard R. Muntz. Bounds for quasi-lumpable markov chains. *Elsevier Perf. Eval.*, 20(1-3):223–243, 1994.
- [76] G. Franks, T. Al-Omari, M. Woodside, O. Das, and S. Derisavi. Enhanced modeling and solution of layered queueing networks. *Software Engineering, IEEE Transactions on*, 35(2):148–161, 2009.
- [77] L. Gallina, G. Dei Rossi, A. Marin, and S. Rossi. Evaluating resistance to jamming and casual interception in mobile wireless networks. In *Proc. of Int. Conf. MSWIM*, pages 151–158, Paphos, CY, 2012. ACM.
- [78] E. Gelenbe. Product form networks with negative and positive customers. *J. of Appl. Prob.*, 28(3):656–663, 1991.
- [79] E. Gelenbe. G-networks with triggered customer movement. *J. of Appl. Prob.*, 30:742–748, 1993.
- [80] E. Gelenbe and I. Mitrani. *Analysis and Synthesis of Computer Systems*. Academic Press, New York, 1980.
- [81] S. Gilmore and J. Hillston. The PEPA Workbench: A tool to support a process algebra based approach to performance modelling. In *Proc. of Seventh Int. Conf. on Modelling Techniques and Tools for Comp. Perf. Eval., LNCS 794*, pages 353–368, Vienna, 1994. Springer.
- [82] S. Gilmore and J. Hillston. The PEPA Workbench: A Tool to Support a Process Algebra-based Approach to Performance Modelling. In *Proc. of the Seventh International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*, number 794 in LNCS, pages 353–368, Vienna, May 1994. Springer.

- [83] S. Gilmore, J. Hillston, and M. Ribaudó. An Efficient Algorithm for Aggregating PEPA Models. *IEEE Trans. on Software Eng.*, 27(5):449–464, 2001.
- [84] W. J. Gordon and G. F. Newell. Cyclic queueing networks with exponential servers. *Operations Research*, 15(2):254–265, 1967.
- [85] W. J. Gordon and G. F. Newell. Cyclic queueing networks with restricted length queues. *Operations Research*, 15(2):266–277, 1967.
- [86] M. Gribaudo, D. Codetta-Raiteri, and G. Franceschinis. Draw-net, a customizable multi-formalism, multi-solution tool for the quantitative evaluation of systems. In *Quantitative Evaluation of Systems, 2005. Second International Conference on the*, pages 257–258. IEEE, 2005.
- [87] P. Harrison and J. Hillston. Exploiting quasi-reversible structures in Markovian process algebra models. *The Computer Journal*, 38(7):510–520, 1995.
- [88] P. G. Harrison. Reversed processes, product forms, non-product forms and a new proof of the BCMP theorem. In *Int. Conf. on the Numerical Solution of Markov Chains (NSMC 2003), Urbana IL, USA, September 2-5 2003*, pages 289–304, September 2003.
- [89] P. G. Harrison. Turning back time in Markovian process algebra. *Theoretical Computer Science*, 290(3):1947–1986, 2003.
- [90] P. G. Harrison. Compositional reversed Markov processes, with applications to G-networks. *Perf. Eval., Elsevier*, 57(3):379–408, 2004.
- [91] P. G. Harrison. Reversed processes, product forms and a non-product form. *Linear Algebra and Its Applications*, 386:359–381, July 2004.
- [92] P. G. Harrison and T. T. Lee. Separable equilibrium state probabilities via time reversal in Markovian process algebra. *Theoretical Computer Science*, 346(1):161–182, 2005.
- [93] Peter G. Harrison and Ben Strulo. SPADES - a Process Algebra for Discrete Event Simulation. *Journal of Logic and Computation*, 10(1):3–42, January 2000.
- [94] A. Heindl. Decomposition of general queueing networks with mmpp inputs and customer losses. *Perf. Eval.*, 51(1-4):117–136, 2003.
- [95] W. Henderson, D. Lucic, and P. G. Taylor. A net level performance analysis of Stochastic Petri Nets. *J. Austral. Math. Soc. Ser. B*, 31:176–187, 1989.
- [96] W. Henderson and P. Taylor. Product form in networks of queues with batch arrivals and batch services. *Queueing Systems*, 6:71–88, 1990.

- [97] W. Henderson and P. Taylor. Some new results on queueing networks with batch movements. *J. of Applied Prob.*, 28:409–421, 1990.
- [98] H. Hermanns, U. Herzog, and J. P. Katoen. Process algebra for performance evaluation. *Theor. Comput. Sci.*, 274(1-2):43–87, 2002.
- [99] H. Hermanns, U. Herzog, and V. Mertsiotakis. Stochastic process algebras: between LOTOS and Markov chains. *Comp. Netw. and ISDN Syst*, 30:901–924, 1998.
- [100] Holger Hermanns. *Interactive Markov chains: and the quest for quantified quality*. Springer-Verlag, 2002.
- [101] J. Hillston. *A Compositional Approach to Performance Modelling*. PhD thesis, Department of Computer Science, University of Edinburgh, 1994.
- [102] J. Hillston and N. Thomas. Product form solution for a class of PEPA models. *Perf. Eval.*, 35(3–4):171–192, 1999.
- [103] C. Hirel, B. Tuffin, and K. Trivedi. SPNP: Stochastic Petri Nets. version 6.0. *Computer Performance Evaluation Modelling Techniques and Tools*, pages 354–357, 2000.
- [104] C. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.
- [105] K. P. Hoyme, S. C. Bruell, P. V. Afshari, and R. Y. Kain. A tree-structured Mean Value Analysis algorithm. *ACM Trans. Comp. Syst.*, 4(2):178–185, 1986.
- [106] M. Iacono, E. Barbierato, and M. Gribaudo. The simthesys multiformalism modeling framework. *Computer & Mathematics with Applications*, 2012.
- [107] J. R. Jackson. Jobshop-like queueing systems. *Management Science*, 10:131–142, 1963.
- [108] M. N. Jacobi. A robust spectral method for finding lumpings and meta stable states of non-reversible Markov chains. *Elect. Trans. on Num. An.*, 37:296–306, 2010.
- [109] A.M. Johnson Jr and M. Malek. Survey of software tools for evaluating reliability, availability, and serviceability. *ACM Computing Surveys (CSUR)*, 20(4):227–269, 1988.
- [110] K. Kant. *Introduction to Computer System Performance Evaluation*. McGraw-Hill, 1992.
- [111] F. Kelly. *Reversibility and stochastic networks*. Wiley, New York, 1979.

- [112] John G. Kemeny and J. Laurie Snell. *Finite Markov Chains*, chapter II. D. Van Nostrand Company, inc., 1960.
- [113] John G. Kemeny and J. Laurie Snell. *Finite Markov Chains*. Springer, second edition, 1976.
- [114] M. Kirschnick. The performance evaluation and prediction system for queueing networks (PEPSY-QNS). Technical Report TR-14-18-94, June 1994.
- [115] L. Kleinrock. *Queueing Systems*, volume 1 (Theory). John Wiley and Sons, 1975.
- [116] Leonard Kleinrock. *Message delay in communication nets with storage*. PhD thesis, Massachusetts Institute of Technology, 1963.
- [117] W. J. Knottenbelt. Generalised Markovian Analysis of Timed Transition Systems. Master's thesis, Department of Computer Science, University of Cape Town, May 1996.
- [118] Samuel Kounev and Christofer Dutz. QPME - A Performance Modeling Tool Based on Queueing Petri Nets. *ACM SIGMETRICS Performance Evaluation Review, Special Issue on Tools for Computer Performance Modeling and Reliability Analysis*, January 2009.
- [119] M. Kwiatkowska, G. Norman, and D. Parker. Prism 4.0: Verification of probabilistic real-time systems. In *Computer Aided Verification*, pages 585–591. Springer, 2011.
- [120] S. Lafon and A. Lee. Diffusion maps and coarse-graining: A unified framework for dimensionality reduction, graph partitioning, and data set parametrization. *IEEE Trans. on Pattern Anal. and Mach. Intell.*, 28:1393–1403, 2006.
- [121] S. S. Lam. Queueing networks with capacity constraints. *IBM Journal of Res. and Dev.*, 21(4):370–378, 1977.
- [122] S. S. Lam. Dynamic scaling and growth behavior of queueing network normalization constants. *J. ACM*, 29(2):492–513, 1982.
- [123] S. S. Lam and Y. L. Lien. A tree-convolution algorithm for the solution of queueing networks. *Commun. ACM*, 26(3):203–215, 1983.
- [124] S. S. Lavenberg. *Computer Performance Modeling Handbook*. Academic Press, New York, 1983.
- [125] A. M. Law and W. D. Kelton. *Simulation modeling and analysis*. McGraw-Hill, 3rd edition, 2000.



- [126] A. A. Lazar and T. G. Robertazzi. Markovian Petri Net Protocols with Product Form Solution. *Perf. Eval.*, 12(1):67–77, 1991.
- [127] E. D. Lazowska, J. L. Zahorjan, G. S. Graham, and K. C. Sevcick. *Quantitative system performance: computer system analysis using queueing network models*. Prentice Hall, Englewood Cliffs, NJ, 1984.
- [128] J. Y. Le Boudec. 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*, pages 78–91, New York, NY, 1986. ACM Press.
- [129] C. Lindemann. Stochastic modeling using dspnexpress. In *Petri Nets and Performance Models, 1995., Proc. of the Sixth International Workshop on*, pages 208–209. IEEE, 1995.
- [130] J. Mairesse and H.-T. Nguyen. Deficiency Zero Petri Nets and Product Form. In *Proc. of the 30th Int. Conf. on App. and Theory of Petri Nets, PETRI NETS '09*, pages 103–122, Paris, France, 2009. Springer-Verlag.
- [131] A. Marin, S. Balsamo, and P. G. Harrison. Analysis of stochastic Petri nets with signals. *Perform. Eval.*, 69(11):551–572, 2012.
- [132] A. Marin and S. Rota Bulò. A general algorithm to compute the steady-state solution of product-form cooperating Markov chains. In *Proc. of MASCOTS 2009*, pages 515–524, London, UK, September 2009.
- [133] A. Marin and M. G. Vigliotti. A general result for deriving product-form solutions of markovian models. In *Proc. of First Joint WOSP/SIPEW Int. Conf. on Perf. Eng.*, pages 165–176, San José, CA, USA, 2010. ACM.
- [134] M. A. Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis. *Modelling with generalized stochastic Petri nets*. Wiley, 1995.
- [135] Moreno Marzolla. The `qnetworks` toolbox: A software package for queueing networks analysis. In Khalid Al-Begain, Dieter Fiems, and William J. Knottenbelt, editors, *Analytical and Stochastic Modeling Techniques and Applications, 17th International Conference, ASMTA 2010, Cardiff, UK, Proc.*, volume 6148 of *LNCIS*, pages 102–116. Springer, 2010.
- [136] John F. Meyer, A. Movaghar, and William H. Sanders. Stochastic activity networks: Structure, behavior, and application. In *International Workshop on Timed Petri Nets*, pages 106–115, Washington, DC, USA, 1985. IEEE Computer Society.
- [137] R. Milner. *Communication and Concurrency*. Prentice-Hall, 1989.

- [138] Robin Milner. *Communicating and mobile systems: the pi calculus*. Cambridge university press, 1999.
- [139] A. Miner and G. Ciardo. Efficient reachability set generation and storage using decision diagrams. In *LNCS 1639, Application and Theory of Petri Nets 1999*, pages 691–691. Springer, 1999.
- [140] Andrew S. Miner, Gianfranco Ciardo, and Susanna Donatelli. Using the exact state space of a markov model to compute approximate stationary measures. In *Proc. of ACM SIGMETRICS*, pages 207–216, New York, NY, USA, 2000. ACM.
- [141] M. K. Molloy. Performance analysis using stochastic petri nets. *IEEE Trans. on Comput.*, 31(9):913–917, 1982.
- [142] T. Murata. Petri nets: Properties, analysis and applications. *Proc. of the IEEE*, 77(4):541–580, 1989.
- [143] R. Nelson. The mathematics of product-form queueing networks. *ACM Computing Survey*, 25(3):339–369, 1993.
- [144] J. F. Pérez, J. Van Velthoven, and B. Van Houdt. Q-mam: a tool for solving infinite queues using matrix-analytic methods. In *Proc. of the 3rd Int. Conf. on Performance Evaluation Methodologies and Tools, ValueTools '08*, pages 16:1–16:9, Brussels, Belgium, 2008. ICST.
- [145] D. C. Petriu, J. E. Neilson, C. M. Woodside, and S. Majumdar. Software bottlenecking in client-server systems and rendezvous networks. *IEEE Trans. on Software Eng.*, 21(9):776–782, 1995.
- [146] B. Plateau. On the stochastic structure of parallelism and synchronization models for distributed algorithms. *SIGMETRICS Perform. Eval. Rev.*, 13(2):147–154, 1985.
- [147] R.J. Pooley. The integrated modelling support environment. In *Proc. of the 5th International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*, pages 1–16, 1991.
- [148] C. PRIAMI. Stochastic  $\pi$ -calculus. *Computer journal*, 38(7):578–589, 1995.
- [149] M. Raiser. Mean Value Analysis and Convolution method for queue-dependent servers in closed queueing networks. *Perform. Eval. Elsevier*, 1(1):7–18, 1981.
- [150] S. Ramani and K. Trivedi. Srept: software reliability estimation and prediction tool. *Computer Performance Evaluation. Modelling Techniques and Tools*, pages 358–361, 2000.

- [151] M. Resiser and S. S. Lavenberg. Mean Value Analysis of closed multichain queueing network. *J. ACM*, 27(2):313–320, 1980.
- [152] J.A. Rolia and K.C. Sevcick. The methods of layers. *IEEE Trans. on Software Eng.*, 21(8):682–688, 1995.
- [153] S. M. Ross. *Stochastic Processes*. John Wiley & Sons, 2nd edition, 1996.
- [154] Sheldon M. Ross. *Introduction to Probability Models*. Academic Press, ninth edition, 2006.
- [155] Ramin Sadre and Boudewijn Haverkort. Fifiqueues: Fixed-point analysis of queueing networks with finite-buffer stations. In Boudewijn Haverkort, Henrik Bohnenkamp, and Connie Smith, editors, *Computer Performance Evaluation Modelling Techniques and Tools*, volume 1786 of *LNCS*, pages 324–327. Springer, 2000.
- [156] R. Sahner, K. Trivedi, and A. Puliafito. *Performance and Reliability Analysis of Computer Systems - An Example-Based Approach Using the SHARPE Software Package*. Kluwer Academic Publishers, Boston, M.A., 1996.
- [157] C. H. Sauer. Computational algorithms for state-dependent queueing networks. *ACM Trans. Comp. Syst.*, 1(1):67–92, 1983.
- [158] C. H. Sauer and K. M. Chandy. *Computer Systems performance modeling*. Prentice-Hall, Englewood Cliffs, 1981.
- [159] M. Sereno. Towards a product form solution for stochastic process algebras. *The Computer Journal*, 38(7):622–632, December 1995.
- [160] M. Sereno and G. Balbo. Mean Value Analysis of stochastic Petri nets. *Perform. Eval. Elsevier*, 29(1):35–62, 1997.
- [161] C. Smith and L. Williams. Software performance engineering. *UML for Real*, pages 343–365, 2004.
- [162] C.U. Smith and L.G. Williams. Performance and scalability of distributed software architectures: An spe approach. *Parallel and Distributed Computing Practices*, 3(4), 2002.
- [163] G.W Stewart. Computable error bounds for aggregated Markov chains. *J. of the ACM*, 30(2):271–285, 1983.
- [164] W. J. Stewart. *Introduction to the Numerical Solution of Markov Chains*. Princeton University Press, UK, 1994.
- [165] W. J. Stewart. *Probability, Markov Chains, Queues, and Simulation*. Princeton University Press, UK, 2009.

- [166] R. Suri. Robustness of queuing network formulas. *J. ACM*, 30(3):564–594, 1983.
- [167] H. M. Taylor and S. Karlin. *An Introduction To Stochastic Modeling*. Academic Press, 3rd edition, 1998.
- [168] D. Towsley. Queuing network models with state-dependent routing. *J. ACM*, 27(2):323–337, 1980.
- [169] M. Tribastone, A. Duguid, and S. Gilmore. The PEPA eclipse plugin. *ACM SIGMETRICS Performance Evaluation Review*, 36(4):28–33, 2009.
- [170] K. S. Trivedi. *Probability and statistics with reliability, queuing and computer science applications*. Wiley-Interscience, second edition, 2002.
- [171] K.S. Trivedi and R. Sahner. Sharpe at the age of twenty two. *ACM SIGMETRICS Performance Evaluation Review*, 36(4):52–57, 2009.
- [172] S. Tucci and C. Sauer. The tree MVA algorithm. *Perform. Eval. Elsevier*, (5(3)):187–196, August 1985.
- [173] N. van Dijk. *Queueing networks and product forms*. John Wiley, 1993.
- [174] M. Veran and D. Potier. Qnap2: A portable environment for queueing system modelling. In *Proc. of the International Conference on Modelling Techniques and Tools for Performance Analysis*, pages 5–24, 1984.
- [175] C.M. Woodside, J. Neilson, S. Petriu, and S. Mjumdar. The stochastic rendezvous network model for performance of synchronous client-server-like distributed software. *IEEE Transaction on Computer*, 44:20–34, 1995.
- [176] Rui Xu and II Wunsch, D. Survey of clustering algorithms. *IEEE Trans. on Neural Networks*, 16(3):645–678, 2005.
- [177] A. Zimmermann, J. Freiheit, R. German, and G. Hommel. Petri net modelling and performability evaluation with timenet 3.0. In *TOOLS '00: Proc. of the 11th Int. Conf. on Computer Performance Evaluation: Modelling Techniques and Tools*, pages 188–202, London, UK, 2000. Springer-Verlag.