Università
Ca'Foscari
Venezia

DOTTORATO DI RICERCA
IN INFORMATICA

CICLO **XXX**
A.A. 2014/2015 - 2016/2017

TESI DI RICERCA

Product-Forms beyond Quasi-Reversibility

SETTORE SCIENTIFICO DISCIPLINARE DI AFFERENZA: INF/01

COORDINATORE DEL DOTTORATO
Prof. Riccardo FOCARDI

SUPERVISORE
Prof. Sabina ROSSI
Prof. Andrea MARIN

DOTTORANDO
Filippo CAVALLIN
Matricola 840031

Contents

Abstract

Assessing the performance and reliability of computer and telecommunication systems requires the development of stochastic models whose state spaces are very large. This problem is often known as *State Space Explosion*. As a consequence, general purpose algorithms for their solution cannot be applied straightforwardly. This problem may be found both in continuous or discrete time. To tackle this problem we resorted to the theory of product-forms, including the latest theoretical developments in the field such as the Reversed Compound Agents Theorem (RCAT) and new forms of time-reversibility. The main contribution of our work consists in the identification of classes of product-form models that are not captured by previous results. More specifically, our contribution can be summarized as follows:

- We identified a process algebraic specification of models including instantaneous propagation of signals in continuous time such as those required to describe the G-networks with negative customers and triggers. As an application of this result we introduced an original model which allows one to perform an exact analysis for a class of cache systems based on the Time-To-Live (TTL) policy with resets.
- We characterized a class of models suitable for the quantitative analysis of reversible computations. We showed that our results can be useful for the performance evaluation of speculative distributed simulations.
- Finally, we analysed product-form models also in discrete time and provided a product-form formulation for the Probabilistic Input/Output Automata (PIOA).

Chapter 1

Introduction

1.1 Preface

This thesis consists of two parts. The **first part** introduces the formalisms used and state of the art in the fields of product-form, time-reversibility and probabilistic process algebras. We present also the main results on product-forms and a brief description of some formalisms in both continuous (i.e., PEPA and Stochastic Automata) and discrete (i.e., PIOA) time. The second part illustrates the novel contributions of this work.
Chapter 2 introduces the Markovian stochastic models, in particular Markov chains and their applications to computer systems. We focus on Markovian queueing networks (QN) giving their definition and two main examples: $M/M/.$ queuing networks (i.e., Jackson's networks) and G-Networks. We give a description of both *basic G-Network* and all its extensions.
The last introductory part is Chapter 3 in which we describe compositional modelling. We illustrate Markovian process algebras and their main results for the product-forms. In the first part we focus on continuous time models with the Performance Evaluation Process Algebra (PEPA) and with Stochastic Automata (SA). After a brief description of discrete time process algebras, we recall the Probabilistic Input/Output Automata (PIOA) and its formal description. In the last part of this chapter, we examine product-forms, and special attention is devoted to the presentation of the RCAT theorem and the notion of quasi reversibility which are the basis from which our work starts. We also present the main theorem on product-form QNs (i.e., BCMP theorem) and we illustrate the properties of the queueing model that imply the BCMP product-form. These properties can be expressed in terms of a characterization of the scheduling discipline or in terms of properties of the underlying CTMC (e.g., Local Balance Property or $M \implies M$ property).

The **second part** of the thesis illustrates our original contributions. Chapter 4 presents propagation of signals in continuous time. We describe a case study with G-Networks with signals. In these networks, one or more customers are forced to move to another queue when a signal enters their queue and according to a Markovian routing rule they enter a new queue or leave the network instantaneously. This is particularly useful to model synchronised or triggered motions; e.g., systems in which work and customers can be moved from one queue to another upon the arrival of an external or internal signal. Applications for G-Network models with signals can be found in flow-control in communications systems or to perform load balancing in distributed systems. More complex systems can be analysed and developed with these models and in particular, they are outside the possible solutions of BCMP-networks presented in [14, 54]. We de-

scribe product-form and stationary probability distribution of the case study. We also give a method to represent signals using the PEPA language. In order to do this, we have to introduce an encoding method called Double Index (DI) solution, for modelling G-Networks with triggers using PEPA. This method uses the concept of a double index. The double index in a process can fully trace the information about the state of another process. This increases the "dependence" between the "tracker" process with respect to the "tracked" one, but completely eliminates any possible uncertainty about possible choices of tracker regarding actions of tracked processes. In this way, a process can know exactly how many positive customers are present in the system and where and so it can decide to perform some specific kind of action. Furthermore, it can be also informed whether the departure of a customer will leave that queue empty or not. Additionally, we want to model G-Networks with signals, also satisfying RCAT conditions. Thus, we introduce the notions of *Phantom State* and *Impossible Actions* to satisfy these conditions maintaining the same behaviours at the same time. Impossible actions will never occur due to the fact that they will never cooperate with the corresponding active/passive actions in other processes. In order to do this, the so-called phantom state is added. This state will never be reachable during cooperation because all the actions which lead to it are impossible actions. In this way all the actions from an unreachable state F will be impossible too, because they will never occur thanks to the non reachability of F.
In Chapter 5 we propose a new model for the analysis of systems with ageing objects such as a Time-To-Live cache. We consider a model with an underlying Continuous Time Markov Chain in which objects can be completely or partially rejuvenated. In the former case the object becomes fresh, while in the latter all the objects are simultaneously rejuvenated so that the youngest becomes fresh. We show that under the so-called Independent Reference Model assumption our model is numerically tractable and has a product-form equilibrium distribution. Furthermore, we consider the case in which the object ageing stops after a certain threshold and hence the partial rejuvenation introduces a probabilistic behaviour. Also in this case, we can derive a product-form equilibrium distribution under some mild conditions. The models presented in our work may be interpreted as a new class of G-Networks with catastrophes and partial flushing. It is worthy of note that these models are not quasi reversible and they do not satisfy RCAT conditions.
In Chapter 6 we study reversible computations. Reversible computations have two execution directions: forward, corresponding to the usual notion of computation, and backward that restores previous states of the execution. Various applications and problems related to reversible computations have been widely studied in different research areas and from different viewpoints, including functional analysis and energy consumption. Various formalisms and models have been proposed in the literature to represent and assess qualitative properties of reversible computations such as their correctness or if two reversible processes are equivalent in some terms. Most of the proposed approaches are based on process algebras that do not include any notion of computation time. In this chapter we propose the adoption of Markovian stochastic models to assess the quantitative properties of reversible computations. Under some conditions, we show that the notion of time reversibility for Markov chains can be used to efficiently derive some performance measures of reversible computations. The importance of time-reversibly relies on the fact that, in general, the process's stationary distribution can be derived efficiently by using numerically stable algorithms. We will review the main results about time-reversible Markov processes and discuss how to apply them to tackle the problem of quantitative evaluation of reversible computations.
The last part showing original results is Chapter 7 in which we study product-forms for

models in discrete time. Probabilistic I/O automata (PIOAs) provide a modelling framework that is well suited for describing and analysing distributed and concurrent systems. They incorporate a notion of probabilistic choice as well as a notion of composition that allows one to construct a PIOA for a composite system from a collection of simpler PIOAs representing the components. Differently from other probabilistic models, the local actions of a PIOA are associated with time delays governed by independent random variables with continuous-time exponential distributions. Our contribution consists of studying the product-form property for PIOAs. Our main result is the formulation of a theorem giving sufficient conditions for a composition of PIOAs to be in product-form and hence to efficiently compute its stationary probabilities.
Finally, in Chapter 8, we present the conclusions of our work.

1.2 Introduction of Thesis

Stochastic models are powerful tools for assessing the non functional quantitative properties of computer networks, communication systems, and software architectures. In many practical applications, Markov processes are the stochastic processes underlying the considered models and their performance evaluation is carried out by using the well-known methods for the analysis of transient or stationary behaviour of Markov processes. In the first part of our work in this thesis, we will focus on the analysis of models whose underlying process is a Continuous Time Markov Chain (CTMC). Several higher level formalisms that are widely applied for quantitative analysis are based on Markov processes, including Stochastic Process Algebras (SPA), Stochastic Petri Nets (SPN), Stochastic Automata Networks (SAN) and Queueing Networks (QN). Although the quantitative analysis based on these formalisms can be obtained by the direct solution of the underlying Markov chain, the state space dimension of the process in general grows exponentially with the model dimension. This is known as the state-space explosion problem and becomes intractable from the computational viewpoint as the problem size increases. In order to overcome this problem, various techniques have been proposed in the literature, including the state-space reduction by aggregating (or lumping) methods, approximation techniques, and the identification of product-form solutions for state probabilities of the Markov chain. The product-form theory provides techniques to derive the equilibrium state distribution of a complex model based on the analysis of its components in isolation. Product-form models consist of a set of interacting sub-models whose solutions are obtained by isolating them from the rest of the systems. Then, the stationary state distribution of the entire model is computed as the (normalised) product of the stationary state distributions of the sub-models. Various classes of product-form models have been defined for different formalisms and some of them can be analysed through efficient algorithms with a low polynomial complexity in the model dimension. Product-form has been widely investigated for queueing network models [14, 74]. These product-form models have closed-form expressions of the stationary state distributions that lead to efficient solution algorithms. For more general Markov models and by the compositionality property of Stochastic Process Algebra, the Reversed Compound Agent Theorem (RCAT) [60, 11] provides a product-form solution of a stationary CTMC defined as a cooperation between two sub-processes under certain conditions. This result gives a unified view of most of the commonly used product-forms.

Since its introduction [22], the theory of product-form solutions has been playing an important role for the practical analysis of models with underlying CTMCs as it allows for

an efficient derivation of the stationary performance indices even when the process's state space is huge and the analysis methods based on the solution of the system of global balance equations become computationally prohibitive. Even more interestingly, for a class of product-form models, including the ones we are studying here, the performance indices can be derived without even generating the joint state space. Successful applications of product-form theory include the BCMP theorem [14], the modelling of neural networks [51], the analysis of systems with fork and join constructs [84], the loss networks [76] and the performance evaluation of wireless networks [17], just to mention a few.

In the second part of our work in this thesis, we also study discrete time models. In discrete time we can use probabilistic automata. A major distinction of these automata is that between fully probabilistic and non-deterministic ones. A fully probabilistic automaton gives a probability distribution (over a set of states or states combined with actions) to every choice it can do. The probability distribution captures the uncertainty about the next state. We will obtain a discrete time Markov chain if we abstract away from the actions in a fully probabilistic automaton. Subsequently, we can apply standard techniques to analyse the resulting Markov chains. Sometimes, we cannot represent probabilistically the incomplete knowledge about the system behaviour. In these cases we should consider more than one possible transition. We speak in this case of a non-deterministic probabilistic automaton. Non-determinism is essential for modelling scheduling freedom, implementation freedom, the external environment and incomplete information. Furthermore, non-determinism is essential for the definition of an asynchronous parallel composition operator that allows interleaving. Non-deterministic choices could be external or internal, due to the fact that they are influenced by the environment or by the system itself. We use the term non-determinism for full non-determinism either with internal or external non-deterministic choices.
We can further categorize the automata types, grouping them in several subsections reflecting their common properties. Basically, every type of probabilistic automata arises from the plain definition of a transition system with or without labels. We can add probabilities to every transition, or to transitions labelled with the same action. There can be also a distinction between probabilistic and ordinary (non-deterministic) states, where only the former ones include probabilistic information, or the transition function can be equipped with structure that provides both non-determinism and probability distributions. The two main groups of probability system are the *reactive* and the *generative* models. The probabilities are distributed over the outgoing transitions labelled with the same action in reactive systems or they are distributed over all outgoing transitions from a state in a generative one. Reactive systems wait (*react*) for an external input (either inside or outside the system) for finalize their actions while generative systems start (*generate*) their own actions. A reactive system acts probabilistically by choosing the next state according to a probability distribution when it receives input from the environment and it has no probabilistic assumptions about the behaviour of the environment. On the other hand, a generative system chooses the next transition according to the probability distribution assigned to the state. Thus, when it chooses the next transition, the system moves to another state while generating the output action. Note that in a generative system there is no non-determinism present, while in a reactive system there is only external non-determinism.
In our work, we use a fusion between reactive and generative models: *Input/Output Model*. The model of input/output probabilistic automata, introduced by Wu, Smolka and Stark in [101], exploiting the I/O automata by Lynch and Tuttle [82], presents a com-

bination of the reactive and the generative model. In an I/O automaton for every input action there is a reactive transition. Note that the transition function for inputs is always a function and not a partial function as in the reactive models. Hence each input action is enabled in each state of an I/O probabilistic automaton. The output actions are treated generatively. At most one generative probabilistic transition gives the output behaviour of each state.
In I/O automaton there is a parameter called *delay rate* δ, an aspect from continuous-time systems. if we ignore the 0 delays, (i.e., with no active actions) one gets the reactive model and excluding passive actions, one gets the generative model with a delay rate assigned to each state.
The semantics of I/O automata are an extension of labelled automata with probabilities and we consider one with a delay rate parameter δ and with an underlying discrete time Markov chain (DTMC) as common denominator of a wide set of Markovian discrete process algebra. Also the parallel composition operation has been treated in different ways in the discrete time models: as unique composition, synchronous, asynchronous or partly synchronous and partly asynchronous operator or even strongly relying on the specific structure of the systems. As the classes of probabilistic systems can be divided into three groups depending on whether they show reactive, generative or mixed behaviour; each of them allow in essence a similar definition and investigation of parallel composition.
Also, the probabilistic automata [101] are equipped with a *composition operation* by which a complex automaton can be constructed from simpler components. The model draws a distinction between *active* and *passive* action types, and in forming the composition of automata only active/passive synchronisations are permitted.

Motivation

In this thesis we study product-forms beyond the *limitations of RCAT theorem and quasi reversibility property*. We want to find new classes of product-forms as well as new methods to analyse cooperating systems without solving all their global balance equations. The class of product-forms is wide and includes different kinds of systems. Even if the Reversed Compound Agent Theorem (RCAT) [61] and Extended Reversed Compound Agent Theorem (ERCAT) [63] are very general results for the analysis of product-form stochastic models, they have some limits and do not capture all product-forms models. In fact, some systems have a product-form solution, even if they do not satisfy the conditions of these theorems. In [8] we can see a method for analysing Markov modulated processes as well as other product-form model classes, opportunely formulated in order to apply RCAT theorem and its extensions. These include quasi-reversible queueing networks [76], G-Networks with various types of triggers [62], queueing networks with finite capacity and blocking [6], stochastic Petri nets [10, 84] and others. However, there are some examples of meaningful models whose product-form cannot be analysed by applying these results. As far as we know, the methodology used in RCAT and in its extensions is sufficient but not necessary for product-forms. For now, there is not a general rule for product-form solutions thus we cannot state that RCAT it's not just an incomplete version of the general rule (if it will ever exist) even if its main idea could be used to search for new product-forms.
In general, a system of a tandem of two queues in which the service rate of the second depends on the number of customers of the first does not have a product-form stationary distribution. Moreover, as analysed in [14, 76], it is not considered in the state-dependent

service rate functions. However, under some assumptions, the product-form exists even if it is not derivable by previously known results. This is an example of a product-form Markov modulated process whose stationary distribution cannot be derived via ERCAT. We find three different kinds of product-form not satisfying the quasi reversibility property or the RCAT theorem in three different application field:

1. Systems with Ageing Objects;
2. Reversible computations;
3. Probabilistic Input/Output Automata.

As for RCAT theorem. the first two are product-forms from the continuous time field where we can see more clearly that the conditions are not satisfied although. For what concerns the last one, the fact that it's in discrete distances itself even further from RCAT. It makes us notice that there are an entire group of product-forms in discrete time that are not tracked by RCAT and its extensions.
We will give now very brief descriptions of these three application fields and then a summary of our contributions.

Systems with Ageing Objects

We focus on modelling and analysing systems with ageing objects by means of product-form models. These systems consist of a set of objects which are associated with an age (e.g., the time-stamp of their creation or latest access). As time passes, the objects become older. Two types of events can rejuvenate the objects:

- total rejuvenation, i.e., the object timestamp is set to the current time. This event affects only one object.
- partial rejuvenation. In this case, the event affects the whole system since the objects are all rejuvenated for the same time interval so that the youngest is associated with the current timestamp.

An example of such a system is a TTL cache in which the total rejuvenation occurs when an object is accessed and the partial rejuvenation can be seen as a method to prevent an under utilisation of the cache memory in case of periods of inactivity. The networking research community has renewed its interest in the performance of caching systems due to the new delivery methods for distributing contents in the networks. The huge number of proxy servers has led to the design of Content Delivery Networks (CDN) which are used by the content providers to deliver information to a large and dispersed population of users. Caching contents that have the greatest demand closer to the users' locations allows one to improve the client-perceived experience, to reduce the server load and optimize the bandwidth requirements. In this perspective, the caching system plays a fundamental role in the gradual shift from the traditional paradigm of host-to-host communication to the new host-to-content model. Other applications of ageing systems are shown in [53, 56] where the failure of nodes in distributed systems are handled by means of checkpoints.

The study in this field led us to the publication of the paper:
A Product-Form Model for the Analysis of Systems with Aging Objects, (F. Cavallin, A. Marin, S. Rossi; *Proc. of Int. Conf. MASCOTS 2015; pp. 136-145*) [24].

Reversible computations

Reversible computations have two execution directions: forward, corresponding to the usual notion of computation, and backward that restores previous states of the execution. Various applications and problems related to reversible computations have been widely studied in different research areas and from different viewpoints, including functional analysis and energy consumption (e.g., [80, 96] and the references therein). Also the RCAT characterisation of product-form solutions is connected to time-reversibility: the solution is based on the definition of a set of transition rates in the time-reversed process. Further notions of reversibility have been introduced in [110, 76] for dynamically reversible processes where some states of the direct and reversed processes are interchanged, and more recently the ρ-reversibility for reversible processes with arbitrary state renaming [87, 86]. Some results on properties and product-form solutions have been recently derived for this class of time-reversibility [89]. Various formalisms and models have been proposed in the literature to represent and assess qualitative properties of reversible computations such as their correctness or if two reversible processes are equivalent in some terms. Most of the proposed approaches are based on process algebras that do not include any notion of computation time [31, 80]. We focus on the quantitative analysis and evaluation of reversible computations based on Markov stochastic processes. The dynamic behaviour of the forward and backward computation may be represented by stochastic models that include the notion of time. Hence, under certain conditions, time-reversibility of stochastic processes can be applied to assess quantitative properties of reversible computations.
Quantitative models based on Markov processes have been widely applied for the analysis and evaluation of complex systems (see e.g., [44, 21]). Markov models and formalisms have the advantage of efficient methods and algorithms for studying their behaviour. In particular, under appropriate stationary conditions, one can derive the equilibrium state distribution of a continuous-time Markov chain by applying algorithms with polynomial time complexity in the process state space cardinality [103].
The concept of time-reversibility of Markov stochastic processes has been introduced and applied to the analysis of Markov processes and stochastic networks by Kelly [76]. A reversible Markov process has the property that when the process obtained by reversing the direction of time is formed, it has the same probabilistic behaviour of the original one. Early applications of these results led to the characterisation of product-form solutions for some models with underlying time-reversible Markov process, such as closed exponential Queueing Networks [14, 58].

The study in this field led us to the publication of the paper:
Applying reversibility theory for the performance evaluation of reversible computations, (M.S. Balsamo, F. Cavallin, A. Marin, S. Rossi; *Proc. of Int. Conf. ASMTA 2016; pp. 45-59*) [5].

Probabilistic Input/Output Automata

Probabilistic Input/Output automata (PIOAs) have been introduced in [102, 101] as a formalism aimed at modelling distributed and concurrent systems in a compositional way. However, the interest in their application goes beyond the pure engineering applications [13]. PIOAs incorporate a notion of probabilistic choice and time delays for locally controlled actions that distinguish them from earlier work [82]. The definition of formalisms for modelling probabilistic systems has been extensively investigated in the literature

both in the field of process algebras and automata theory. One of the key factors that characterises the proposed methodologies is clearly the semantics of the composition. Giving a reasonable way of composing probabilistic systems is challenging because the probabilities that are specified within each single component have a "loca" meaning. In general, they are not sufficient to describe the probabilistic behaviour of the joint model without further assumptions such as the time scale. In the PIOA model, this problem is solved by associating an exponentially distributed delay parameter with each state. Intuitively, a PIOA first draws a random delay time from an independent exponentially distributed random variable and then performs the probabilistic choice. Therefore, in the composition of a collection of PIOAs, the usual *race condition policy* used in [70, 97] is applied. Another feature of PIOAs is the way they handle synchronisations, which is inspired by the I/O automata originally defined in [82]. PIOAs communicate via input and output actions and can perform internal non-communicating transitions. The communication is seen as a message transmitted on a labelled channel (that we call synchronisation label) by the output automaton. The synchronising automaton can read the message and perform a probabilistic transition accordingly. For each PIOA the sum of the probabilities associated with output and internal transitions, called locally controlled transitions, must be 1. On the other hand, upon the reception of a message, the PIOA immediately reacts, i.e., the sum of the probabilities associated with the message-receiving transitions outgoing from each state of a PIOA must be 1 for each of them separately (including the possible self-loops).
PIOAs share with many other formalisms for the quantitative analysis of computer systems the property of having an underlying Markov process that describes the model evolution, and the problem of the exponential growth of the cardinality of the state spaces which makes the derivation of the quantitative indices unfeasible even for relatively small systems. The problem of defining compositional approaches to the quantitative analysis of PIOAs has been addressed in [102] for what concerns the transient behaviour. To the best of our knowledge, the problem of defining a compositional approach for studying the stationary behaviour of PIOAs remains open. In the literature of queueing networks this problem is often associated with the so-called *product-form* analysis which is described in [76] and then extended in numerous subsequent works (see, e.g., [46, 43, 11]). There were some product-form results for Markovian process algebras in the previous decade [61, 66]. Informally, a product-form model can be studied without constructing the stochastic process underlying the composition of the simpler components forming the systems, but these can be studied in isolation. Hence, the computational effort required to compute the stationary quantitative indices is highly reduced.

The study in this field led us to the publication of the paper:
Product-forms for Probabilistic Input/Output Automata, (F. Cavallin, A. Marin, S. Rossi; *PProc. of Int. Conf. MASCOTS 2016; pp. 361-366*) [25].

Main contributions

In this thesis, the main contributions with respect to the literature are the following:

- We analysed the state of art for what concern the analysis and detection of product-forms. We looked for cases not found by well-known methods. We also studied and analysed product-forms in discrete time and their relations with the ones in continuous time. In the last part, we focused our efforts in the study of properties to

detect product-forms directly in discrete time without passing through continuous time.

- We present two models in product-form for the performance evaluation of **systems with ageing objects**. The main difference between the two models is that one allows the object age to grow indefinitely, while the other introduces a maximum age threshold. We discuss the implications for practical applications with some examples. The product-form analysis that we demonstrate is interesting for at least two aspects. The first is that neither the joint CTMC nor the CTMC underlying a single model are reversible as it happens, e.g., in Jackson's queues [74] and G-queues [48]. The second interesting aspect is that synchronisations among objects are not pairwise, i.e., at a given epoch more than two objects can simultaneously change their states. There are few results in this direction in the literature of product-forms. In [50, 38] the authors consider queueing networks in which the departure of a customer from a queue causes a movement of one job from a second queue to a third one, hence causing the simultaneous state change of three components. However, the extension of the result to more than three components is not trivial mainly because the proof technique adopted in those papers is based on solving the system of global balance equations (GBEs). The first model we propose is in the style of G-Networks as proposed in [41], while the one with maximum ageing is, to the best of our knowledge, very peculiar since very few product-forms are known for finite state space models [3, 6]. The contribution of the unbounded model with respect to [41] is twofold. First, the proof is not based on the solution of the system of global balance equations of the joint model. Secondly, we consider individual jumps of the objects to the zero state. Our proof method is based on the quasi-reversibility property [76] and the Reversed Compound Agent Theorem (RCAT) [61, 7]. Both these results provide a way to elegantly prove the product-form of a CTMC but they consider only pairwise synchronisations and hence they cannot be straightforwardly applied to study our models. We show that they can still be used by introducing a passage to the limit for a transition rate in a similar fashion to what has been done in [27, 65, 84]. Proofs of product-forms based on quasi-reversibility are simple to handle and compositional in the sense that they allow the combination of the models that we study here with others which are known to be quasi-reversible while maintaining the product-form of the equilibrium distribution. As a consequence, heterogeneous networks may be studied without constructing the joint Markov chain.

- We show how to numerically derive the models' performance indices of **systems with ageing objects**, without constructing the joint CTMCs. This is important because the structures of these chains can be complex since the transitions corresponding to partial rejuvenations depend on the global state of the models. The derivation of the performance indices requires us to solve a non-linear system of equations. We propose a fixed point algorithm to tackle this problem and show its efficiency and convergence properties on numerous examples. The system of equations admits a unique positive solution. With respect to [41, 42], we do not require any modification of the network of objects in order to obtain the convergence of the algorithm.

- As an example of **systems with ageing objects**, we apply our model for the analysis of a TTL cache with partial rejuvenation. First, we propose an ideal model, whose

implementation is very expensive, in which a timer is associated with each object despite the fact that it is inside or outside the cache. We study the performance indices under the Independent Reference Model (IRM) assumptions [45, 77, 108]. Then, we consider a model in which we maintain the timers only for the objects inside the cache. The partial rejuvenation of objects outside the cache has a probabilistic effect, i.e., the object may remain outside or can be copied inside the cache according to a Bernoulli trial. We prove that it is possible to obtain exactly the same expected performance indices of the ideal model while maintaining the product-form property. We discuss how it is possible to dynamically set the model's parameters to achieve some performance goals. The analyses of TTL caches, often connected to form networks, have been widely addressed in recent years (see, e.g., [15, 34, 35] and the references therein). In our case study, we consider a simpler situation of a single cache as in [91]. Clearly, the analysis becomes challenging because of the partial rejuvenation signals which aim to avoid the under utilisation of the cache.

- For what concerns **Reversible computations**, we review the main results about time-reversible Markov processes and discuss how to apply them to address the problem of quantitative evaluation of reversible computations. We recall the definition of time reversibility for continuous time Markov processes, the main properties and its application for quantitative analysis. We present an abstract model of continuous time Markov chain for representing and performance evaluating reversible parallel computations. Taking advantage of the process reversibility, the stationary distribution of the model can be efficiently derived by using numerically stable algorithms. In particular, we present some product-form results of reversible synchronising automata by applying ρ-reversibility to the underlying Markov process.

- Finally for **PIOAs** our contribution consists in studying the product-form property for PIOAs. Our main result is the formulation of a theorem giving sufficient conditions for a composition of PIOAs to be in product-form and hence to efficiently compute the stationary probabilities.

Chapter 2

Markov chains and applications to computer systems

2.1 Introduction

In this chapter we will introduce the Markovian stochastic models characterized by a stochastic process. By studying these processes, one can analyse a set of model properties such as liveness, performance parameters and model checking issues. In particular we will focus on Markov chains and their applications to computer systems. We focus on the Markovian queueing network (QN) giving its definition and two main examples: $M/M/.$ queues and *G-Networks*. We give a description of both "pure" G-Network and all its derivations.

2.2 Markov Stochastic Models

We introduce in this chapter some notions about Markovian stochastic models. First of all we have to recall what is a stochastic model.
A *mathematical model* is the **quantitative** description of a natural phenomenon, judged using a factor. If a model is characterized by having a stochastic *process* within it, it's called a Stochastic Model. A stochastic process is a family of random variables X_t, where t is a parameter contained in a set T (we can also use it as $X(t)$). The set of random variables are usually defined over the same probability space and indexed by the parameter t. Usually t is a discrete index of a set $T = \{0, 1, 2, 3...\}$. For example, $X(t)$ could represent the outcome of successive dice throws or, more generally, repetitive observations during different times of some characteristics of a system. t could also represent a distance from an arbitrary origin 0, not just time, i.e., $X(t)$ can be the number of people in the interval $(0, t]$ on a sidewalk.
Stochastic processes are characterized by:

- index set of T
- *state space*: the range of all possible value that the random variables $X(t)$ can have
- dependence relations between all the random variables $X(t)$

Both the state space and the index set can be continuous or discrete. A lot of properties such *liveness*, *performance parameters* and *model checking issues* (e.g. deadlocks detection, unreachable states,..) can be derived studying these processes.

Let $X(t)$ be a stochastic process taking values in a state space $\mathcal{S}$ for $t \in \mathbb{T}$. We can have two cases:

- the case of continuous time (CT): the process is called continuous-time stochastic process and therefore t is continuous: $\mathbb{T}$ is the real line $\mathbb{R}$;
- the case of discrete time (DT) stochastic processes, $\mathbb{T}$ is the set of integers $\mathbb{Z}$.

A discrete-time process is usually denoted by X_n with $n \in T$

For what concerns the state space it can be:

- *discrete*: the process is called discrete-space process or just *chain*
- *continuous*: the process is called continuous-space process

We denote the probabilistic behaviour of a stochastic process as:

$$Pr\{X(t_1) \leq x_1; X(t_2) \leq x_2; ...; X(t_n) \leq x_n\}$$

where x_i is an element of the state space. This behaviour is defined by the joint probability distribution function of the random variables $X(t_i)$ for any set $t_i \in \mathbb{T}$, $1 \leq i \leq n$ with $n \geq 1$.
The stochastic process $X(t)$ is said to be *stationary* if $(X(t_1), X(t_2), \ldots, X(t_n))$ has the same distribution as $(X(t_1 + \tau), X(t_2 + \tau), \ldots, X(t_n + \tau))$ for all $t_1, t_2, \ldots, t_n, \tau \in \mathbb{T}$.

2.3 Markov Chains

A stochastic process $X(t)$ is a *Markov Process* if for all $t_1 < t_2 < \cdots t_n < t_{n+1}$ the joint distribution of $(X(t_1), X(t_2), \ldots, X(t_n), X(t_{n+1}))$ is such that

$$P(X(t_{n+1}) = i_{n+1} \mid X(t_1) = i_1, X(t_2) = i_2, \ldots, X(t_n) = i_n) = P(X(t_{n+1}) = i_{n+1} \mid X(t_n) = i_n).$$

The equation can be written more concisely as:

$$P(i_{n+1} \mid i_1, i_2, \ldots, i_n) = P(i_{n+1} \mid i_n).$$

and is valid whenever the conditioning event $(i_1, i_2, \ldots, i_n)$ has positive probability. In other words, for a Markov process its past evolution until the present state does non influence the conditional (on both past and present states) probability distribution of future behaviour. A Continuous-Time Markov Chain (CTMC) or a Discrete-Time Markov Chain (CTMC) is a Markov process with a discrete state space $\mathcal{S}$.
We have to notice that, thanks to the Markov property, the residence time of the process in each state is distributed according to:

- the geometric distribution in the discrete-time case
- the negative exponential distribution in the continuous-time case

A Markov process is *time homogeneous* if the conditional probability $P(X(t+\tau) = j \mid X(t) = i)$ does not depend upon t, and we use the notation, with $i, j \in \mathcal{S}$:

$$CTMC : \lim_{\tau \to 0} \frac{P(X(t+\tau) = j \mid X(t) = i)}{\tau} = q_{ij}$$

$$DTMC : P(X(t+1) = j \mid X(t) = i) = p_{ij}$$

In the continuous case, the transition rate between two states i and j is denoted by q_{ij}, with $i \neq j$. This means that the Markov process in state i remains in that state for a length of time which is exponentially distributed with parameter:

$$q_i = \sum q_{i,j}$$

and when it leaves state i, it moves to state j with probability:

$$p_{q_{ij}} = \frac{q_{i,j}}{q_i}$$

The infinitesimal generator matrix $\mathbf{Q}$ of a Markov process is such that the q_{ij}'s are the off-diagonal elements while the diagonal elements are formed as the negative sum of the extra diagonal elements of each row.
In the discrete case, the value p_{ij} denotes the probability that the chain, whenever in state i, next makes a transition into state j, and is referred to as *one-step transition probability*. The square matrix $\mathbf{P} = (p_{ij})_{i,j \in S}$ is called *one-step transition matrix*, and since when leaving state i the chain must move to one of the states $j \in S$, each row sums to one (e.g., forms a probability distribution): for each $i \in S$

$$\sum_{j \in S} p_{ij} = 1.$$

The n*-step transition probabilities* of the chain are given by

$$p_{ij}^n = P(X(t+n) = j \mid X(t) = i)$$

for $t, n \in \mathbb{Z}$ and $i, j \in S$, denoting the probability that a process in state i will be in state j after n additional transitions.
A state j is said to be *accessible* from state i if there exists a path from i to j ($p_{ij}^n > 0$ in the discrete case for some $n \geq 0$). Two states i and j accessible to each other are said to *communicate* and to belong to the same *equivalence class*. A Markov chain is said to be *irreducible* if there is only one equivalence class, i.e., if all states communicate with each other. The recurrence time T_i of a Markov chain is defined as $T_i = \min\{n \geq 1 \mid X(t_{n+1}) = i \text{ given } X_n = i\}$ (notice that T_i is a random variable.) A state i is said to be *recurrent* if $P(T_i < \infty) = 1$, i.e., the probability that the process will eventually return to the same state is one. Otherwise, it is called *transient*. The mean recurrence time M_i of state i is defined as $M_i = E[T_i]$. A recurrent state i is called *positive recurrent* if $M_i < \infty$, i.e the expected number of steps until the process returns to it is finite. A Markov chain is called *positive recurrent* if all of its states are positive recurrent. Only in a DTMC, a state i is said to have a *period* $d \geq 1$ if d is the greatest integer such that $p_{ii}^n = 0$ if n is not a multiple of d. A state with period 1 is said to be *aperiodic* and a Markov chain is said to be *aperiodic* if all states are aperiodic.
A Markov Chain is said *ergodic* if it is irreducible and positive recurrent (and also aperiodic in the discrete case). For the continuous case only, if the chain is finite, irreducibility is sufficient for ergodicity.
An ergodic Markov Chain possesses a *equilibrium* (*stationary* or *steady-state*) *distribution*, that is the *unique* vector $\boldsymbol{\pi}$ of positive numbers π_i with $i \in \mathcal{S}$ such that

$$\lim_{t \to \infty} P(X(t) = i \mid X(0) = j) = \pi_i \,. \tag{2.1}$$

The non-trivial vector of real numbers $\boldsymbol{\pi}$ satisfying the system of global balance equations (GBEs) is [78]:

$$CTMC : \boldsymbol{\pi}\mathbf{Q} = \mathbf{0} \tag{2.2}$$

$$DTMC : \boldsymbol{\pi} = \boldsymbol{\pi}\mathbf{P} \tag{2.3}$$

and it is called the *invariant measure* of the Markov Chain. If the components of π_i sum to unity,

$$\sum_{i \in \mathcal{S}} \pi_i = 1 \,,$$

then π is a probability distribution on $\mathcal{S}$ and is called steady-state distribution. From the context, it will be clear if π denotes the steady-state distribution or an invariant measure of $X(t)$. Each π_i can be interpreted as the average proportion of time spent by the chain $X(t)$ in state i. A Markov chain with a stationary distribution is said to be in *steady state*. In other words, a Markov chain is in steady-state distribution if the probability distribution of time spent in each state can be computed (i.e., it doesn't go to infinite). Of all the invariant measures of the chains, the only one that is a probability distribution (if it exists) is the stationary distribution. Notice that for irreducible, positive recurrent and periodic DTMCs, although the limiting distribution does not exist, the system (2.3) still admits a unique solution summing to unity that is still called stationary distribution but does not clearly correspond to the limiting distribution. Markov Chains have been widely used in the performance evaluation field in order to derive several model properties (for example [76, 78, 29, 32, 105]).

2.3.1 Examples of Markov Chain Models

A large number of natural, biological and economic phenomena can be described with Markov Chains. The importance of Markov Chains is enhanced by their predisposition for quantitative manipulation.

Stock Model

In a warehouse, goods are consumed during each period of time t according to a random variable μ_t. At the end of each period $(0, 1, ..)$, we check if the remaining amount of goods are under some quantity s. If this is the case, we refill the warehouse to an optimal value S.
According to these rules of warehouse policy, the stock levels of two consecutive periods follow the relation:

$$X_{n+1} = \begin{cases} X_n - \mu_{n+1} & \text{if } s \leq X_n \leq S, \\ S - X_{n+1} & \text{if } X_n \leq s. \end{cases} \tag{2.4}$$

where μ_i is the amount of goods consumed during period i.
Assuming that each consecutive demand $\mu_1, \mu_2, ..$ are independent random variables, the stock quantities $X_0, X_1, ..$ compose a Markov Chain. Its probability matrix can be calculate as:

$$\begin{aligned} P_{ij} &= \Pr\{X_{n+1} = j | X_n = i\} \\ &= \begin{cases} \Pr\{\mu_{n+1} = i - j\} & \text{if } s \leq i \leq S, \\ \Pr\{\mu_{n+1} = S - j\} & \text{if } i \leq s. \end{cases} \end{aligned} \tag{2.5}$$

Ehrenfest Urn Model

Markov chains can be also used to model the diffusion of molecules through a membrane. Let's consider two containers (cells), containing a total of $2n$ balls (molecules), each of them have i and $2n-i$ balls, respectively. We select a random ball from the whole set and move it to the other container. In this way a molecule diffuses at random through the membrane. We can model each selection as a transition of the process.
The molecules fluctuates between the cells with an average drift from the container with the highest concentration to the one with the smaller numbers. Considering X_m the difference between the number of molecules contained in a cell and its mean number n, we have that $\{X_m\}$ is a Markov Chain on the state space $\{-m, -m+1, -m+2, ..., -1, 0, 1, ..., m-1, m\}$ with transition probability computed as follow:

$$P_{ij} = \begin{cases} \dfrac{n-i}{2n} & \text{if } j = i+1, \\ \dfrac{n+i}{2n} & \text{if } j = i-1, \\ 0 & \text{otherwise.} \end{cases}$$

The main focus of this model is the equilibrium distribution of the number of balls in each container.

Markov Chains in Genetics

Markov Chains are also used in genetic fields. We can model the fluctuation of gene frequencies under possible mutations and selections. Assuming that we are studying a fixed population size of $2N$ genes divided in type-A and type-B individuals. The composition of the next generation of individuals is determined by $2N$ independent Bernoulli trials. Considering parents population composed by j A-type and $2N-j$ B-type, then the outcome of each trial could be A (p_j) or B (q_j) with the following probabilities:

$$p_j = \frac{j}{2N}, \quad q_j = 1 - \frac{j}{2N}.$$

Now we can create a Markov chain $\{X_n\}$ where X_n is the number of A-types in the n-th generation. Since the population is constant ($2N$), the space state has $2N+1$ values: $\{0, 1, ..., 2N\}$. The transition probability matrix can be computed with the binomial distribution:

$$\Pr\{X_{n+1} = j | X_n = i\} = P_{ij} = \binom{2N}{j} p_i^j q_i^{2N-j}$$

with $i, j = 0, 1, 2, ..., 2N$.
In this case the end states $0, 2N$ are completely absorbing states, i.e., once X_n is equal to 0 or $2N$ then $X_{n+m} = X_n, \ \forall m > 0$. Under the condition of $X_0 \neq 0$ or $2N$, the main focus here is to compute the probability and the rate of approach of the population to reach the fixation, i.e., that it will become purely composed by only A-genes or B-genes. From this simpler model, we can add other variables (e.g., mutation pressure).

Discrete Customer Service

We can model a customer office in which people arrive and take a place in the waiting line. During each period of time t, an employee can service only one customer, if at

least one is waiting. If no customer is waiting then no service is issued. During each period t, new customers can arrive in the office according to a random variable λ_t, whose distribution is independent with respect to the period which we are considering. The probability of these arrivals are:

$$\Pr\{\text{n customer arrive}\} = \Pr\{\lambda_t = n\} = a_n$$

for $n = 0, 1, 2, .., a_n \geq 0$ and $\sum_{n=0}^{\infty} = 1$.
The state of the system X_t is defined by the number of waiting customers i during the period t. The following period can be computed as:

$$X_{t+1} = \begin{cases} i - 1 + \lambda_t & \text{if } i \geq 1, \\ \\ \lambda_t & \text{if } i = 0, \end{cases}$$

with λ representing the number of new customers arrived in the servicing period.
The focus of this kind of models are important quantities such as the time that the office is idle in the long run (π_0) or the mean waiting time of the customers in the office ($\sum_{k=0}^{\infty}(1+k)\pi_k$).

2.4 Queueing Networks

A queueing system is composed of customers arriving at random times into a queue where they expect some kind of service and then leave the queue. Customers could be whatever can access or be put in a queue demanding for some kind of service. Some examples can be:

- People waiting in a post office to send letters;
- Ships entering a port, waiting for their pier;
- Computer data flowing in a communication system;
- Broken machines waiting to be repaired in a factory

These network models have been extensively used for system performance evaluation and prediction of resource sharing systems such as computer networks, computer and production systems. Queueing networks are powerful and versatile in modelling and analysing such systems. The analysis of queueing systems relies on the theory of stochastic processes [29, 78, 81, 75, 99, 107].
The general description of a queueing network is a set of service centres, which represent resources of the system, that provides some kind of service to a set of waiting customers. Queueing disciplines control the priority of each customer to be served, which competes for the resource service. They possibly wait in the queue if more of them compete for the same resource. We can analyse the system through computation of its performance measures (e.g., utilization, throughput, customer waiting time). A set of random variables that define a stochastic process can be used to describe the dynamic behaviour of a queueing system. Assuming some constraints, we can also define and solve its underlying Markov process to compute its performance values. The relatively high accuracy in the performance measures and the efficiency of model analysis and evaluation lead to the popularity of queueing models for performance analysis. Moreover, the theory of

product-form solutions was decisive. In queueing networks, product-forms have simple and closed-form expressions of the steady-state distribution. This can allow us to compute average performance values with polynomial time complexity in the number of model components with efficient algorithms.
Queueing systems are used to study congestion in computer and communication systems. We can use queueing networks of interacting service centres to model congestion or resource-sharing systems and then compute their performance. The class of queueing networks are very robust models even if we have to make several assumptions about the system. The main solutions to analyse such models are:

- *simulation*: it has a wide application but potentially it can have high development and a high computational cost to obtain accurate estimates. Moreover, these values can be of quite difficult interpretation.
- *analytical methods*: they are based on a set of mathematical relationships characterized by the system behaviour. These methods require that a set of assumptions and constraints are satisfied by the model.

2.4.1 Queueing Systems Definition

The basic model of a queueing network consists in a single service centre which provides its service to the entire system. The theory of stochastic processes is under the analysis of queueing systems and their basic version have been defined in queueing theory to be applied to congestion systems analysis. We can define a continuous-time Markov process associated with the queueing network, assuming that the model random variables are independent and exponentially distributed. In this way we can solve the underlying Markov process to analyse the queueing system.
In Figure 2.1 we can observe a simple queueing system (or service centre).

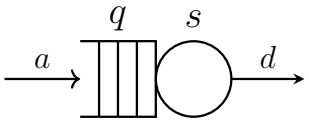


Figure 2.1: Simple Queueing System.

This system models the stream of arriving customers, their waiting in the queue if another customer is being served by the server and then when they receive the service and in the end when they leave the queue.
For example, a simple shop can be modelled by a basic queueing system in which people are customers, their waiting in line is the queue and the employee is the server whose services are tasks of the employee with each person.
We can classify queueing system behaviour, according to five basic characteristics:

1. The arrival process (or input process): is the probability (discrete time) or rate (continuous time) distribution of the customer arrivals. It describes the behaviour of customers arrivals. We can define the time between two consecutive arrivals as a random variable. The average number of arrivals per each unit of time is the *mean*

arrival rate, and is denoted by λ in CT. In DT the probability that some number of customers arrive is denoted by p. We usually assume that the arrival process is a Poisson process and this corresponds to an exponential inter-arrival distribution.

2. The service process: is the description of the behaviour of the customer service. Its the probability (DT) of a service or rate (CT) representing the random time to provide the service to a customer (or group of them in case of batch service). We can define the mean service rate as a random variable μ (CT) and the probability of service as q (DT). In this way, the random variable of the time spent for a customer service is $1/\mu$ and it represents the service time.

3. The queue discipline: is the order in which customers are served. It describes how customers are scheduled in the queue. A customer is forced to wait in the queue if the server is unavailable to provide service to it when it arrives. It has to wait until it can start to receive its service. The problem of a discipline arises when there is more than one waiting customer in the queue. When the server becomes available the discipline has to choose one of the waiting customers to start receiving the service. This kind of customer selection for service is what distinguishes two queueing disciplines.
   Queueing disciplines can be discerned by arrival time, possible services already given to customers and customers priority. If there are no artificial creation or loss of work in the system then the queueing discipline is also work-conserving.
   First Come First Served (FCFS) and Last Come First Served (LCFS) are classical examples of queueing disciplines based only on arrival time.

4. The number of servers: is the set of identical and parallel servers which can provide the service to customers. A common queue is shared by all customers and each server can be a service facility of the system, physically or logically separate.

5. The queue capacity: is the upper limit of the possible number of waiting customers (both in line or receiving service) in the system. Queue can have infinite size (i.e., to have enough slots to receive all arriving customers) or with finite resources, forcing an upper bound to the maximum number of customers that can be present in the queue at the same time. Most analytical computations require the queue to be infinite.

There are many arrival processes that can be found in practice. Two simple and intensively used types are scheduled input and completely random input. In the first, customer arrivals are scheduled at fixed times $(t, 2t, 3t, ..)$, in the latter, customers arrive forming a Poisson process. These two type of arrival process are mathematically tractable and can be used to study more complex cases. A lot of theoretical results can be found if customers arrivals constitute a renewal process. In the exponentially distributed inter-arrival times, the Poisson process of customer arrivals is a special case, even if it is one of the most commonly used. Also the service duration for customers is an independent and identically distributed random variable. In this case, the special case is when all service times have the same fixed duration.
The main target of queueing models is to evaluate a set of system performance measures in terms of more basic quantities. This kind of computation can aid the design process, e.g., computing the benefits/costs of adding another server to the system. Some performance indices are:

- the number of the customers w_p in queue p;

- the number of waiting customer n either waiting in the queue or being served;
- the customer waiting time t_w. Long waiting times to obtain services are annoying for customers in simple queueing networks and they are often the cause of bigger costs in much larger and complex systems;
- the customer response time t_r;
- the system or server utilization U, i.e., the percentage of time in which the system is providing services, therefore busy. In fact, idle servers could create unnecessary costs without any contribution to system performance;
- the system throughput X is one of the main system performance measures and it represents the average number of served customers for each unit of time. In other words, it's the number of customers passing through the system per unit of time in the long run.
- the probability distribution of customers in the system. As we said, queues are not always infinite. In real systems, physical space for new customers has to be planned and provided. With limited slots, a full queue or even with just a large number of waiting customers can adversely affect the arrival process leading to a loss of possible customers. Moreover, the presence of customers in the system can represent a cost for system design.

The most analysed and used measures are the average system throughput and system utilization and the probability distribution.

Considering t_s the service time and s the number of customers in service, we have that the average service time is $E[t_s] = 1/\mu$. Then the following relations hold:

$$n = w + s;$$
$$t_r = t_w + t_s.$$

If we denote the average number of customers in the system $E[n]$ as N and the mean response time $E[t_r]$ as R we have that:

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

We can analyse a queueing system associating a discrete-space stochastic process in which the space state includes the system population. Moreover, we can define an underlying Markov chain, using independent and exponential assumptions. Its stationary solution can be derived using Formula 2.2 (CT) or 2.3 (DT). We can compute other performance measures with the steady-state probability and basic relations (e.g., Little's Law).

2.5 Modelling Examples - M/M/. queues

The Kendall's notation is used to describe queueing processes. A queueing system can be composed by queues defined as $A/B/X/Y/Z/K$, where:

- A: describe the arrival process;
- B: describe the service process;

- X: describe the number of parallel servers;
- Y: describe the capacity of the queue;
- Z: describe the size of the population;
- D: describe the service discipline;

The concise notation $A/B/X$ is more common and implies a queueing system with queues with *infinite capacity*, *infinite population* and *FCFS* queueing discipline. A and B can assume symbols of probability distributions:

- D: deterministic distribution;
- M: exponential distribution;
- G: general distribution.

One of the most common examples is a $M/M/1$ queue that has a Poisson arrival process, exponential service process and a single server. Instead, a $M/G/\infty$ queue has the same arrival process but an arbitrary service distribution and an unlimited number of independent exponential servers.

2.5.1 M/M/1 queues

As we said, this kind of queue has:

- Poisson independent arrivals;
- exponential service time distribution;
- one server with infinite capacity and FCFS discipline.

We denote the arrival rate with λ, the service rate with μ, the traffic intensity with $\rho = \lambda/\mu$ and the system state with n. In this case, its underlying Markov process is a birth and death process with constant rates λ and μ. We can use it to compute the customer population and the stationary state probability π_n. If the stability condition is satisfied by the system,we can obtain:

$$\pi_n = \rho^n(1-\rho) \qquad n \geq 0$$

If the arrival rate is less than the service rate, the $M/M/1$ queue is stable. (i.e., with $\rho < 1$ or $\lambda < \mu$). Furthermore, we can derive other performance measures (average population, mean response time, system throughput and utilization) with the state probability and by Little's law:

$$\begin{aligned} N &= \frac{\rho}{1-\rho}; \\ R &= \frac{1/\mu}{1-\rho}; \\ U &= 1 - \pi_0 = \rho; \\ X &= \lambda. \end{aligned}$$

2.5.2 M/M/m queues

This kind of queue extends basic $M/M/1$ queues with m servers, they have:

- Poisson independent arrivals;
- exponential service time distribution;
- m servers with infinite capacity and FCFS discipline.

As we can see the exponential and independence assumptions remain, the arrival rate is λ, the service rate is μ and system state is n. ρ is now equal to $\lambda/m\mu$. The underlying Markov process is still a birth and death process with constant birth rates λ but with variable state-dependent death rate $\mu_n = \min\{n, m\}\mu$ for all $n \geq 0$. If $\rho < 1$ (i.e., the system satisfies the stability condition) then we can compute π_n as:

$$\pi_n = \begin{cases} \dfrac{(m\rho)^n}{n!}\pi_0 & \text{if } 1 \geq n \geq m, \\ \dfrac{m^m\rho^n}{m!}\pi_0 & \text{if } n > m. \end{cases}$$

where

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

We can now derive other performance measures such average population and mean response time:

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

2.5.3 M/M/∞ queues

This kind of queue has:

- Poisson independent arrivals;
- exponential service time distribution;
- unlimited number of independent exponential server with infinite capacity and FCFS discipline.

The arrival rate is λ, the service rate of each server is μ and system state is n. ρ is now equal to λ/μ.
Here, customers never wait to be served due to the infinite number of servers. In this way, the mean response time for customers is equal to the mean service time of servers: $R = 1/\mu$. The underlying Markov processes associated with these queues are still birth and death processes with constant birth rates λ and variable state-dependent death rate $\mu_n = n\mu$. This time the system is always stable because customers never wait in a queue because they are always served. Hence, the steady-state probability is:

$$\pi_n = \frac{\rho^n}{n!}e^{-\rho} \qquad k \geq 0.$$

We can now compute the average population as $N = \rho$.

A lot of $M/M/1$ queueing systems can be analysed using and solving their underlying Markov chain. For example, $M/M/1/N$ queues with finite capacity N and also $M/M/1//K$ queues with finite population K. We can compute performance measures and the steady-state probability for these kinds of queues and similar basic queueing systems.

2.6 Modelling Examples - G-queues

Another example of the application of the queueing theory are G-Networks [51]. G-Networks, namely *Generalized Queueing Networks* or *Gelenbe Networks*, are open networks composed by G-queues. Erol Gelenbe first introduced them as models for queueing systems and as well as for neural networks. They have specific control functions, which include traffic re-routing and traffic destruction [51, 50, 49].
The foundations of this model are G-queues, which are composed by:

- *Positive Customers*: they arrive from other queues or from outside the system, they wait for service and follow a routing disciplines (like in other conventional network models);
- *Negative Customers*: they arrive from other queues or from outside the system, they don't act as normal customers but they remove (or kill) another waiting customer of the queue (if the queue is non-empty). They represent the need of remove some traffic in case of network congestion. This includes also removing batches of customers;
- *Triggers*: signals arriving from other queues or from outside the system. They can move or kill other customers and displace them in other queues.

Product-form solutions can be used to compute the stationary distribution of G-Networks along with usual quantitative measures. We can exploit this kind of networks to approximate quite general input-output behaviours because of their nature of universal approximator for continuous and bounded functions.

2.6.1 General View

Gelenbe created this new class of queueing networks with two types of customers: positive (or regular) and negative.
Interest, in queueing networks and also in the case of single server node, has been increased since its first introduction of the notion of negative customers. With this theory, significant progress has been made in the versatile class of networks analysis. This progress has helped the development of real application systems like manufacturing, communication and indeed computers. Moreover, they have also enriched the queueing theory.
G-Networks, also known as queueing networks with negative customers, signals, triggers, (...), have the characteristics of containing also the presence of negative customers in contrast with the normal positive ones. If those negative customers arrive in a non-empty queue then there is the witness of a removal of some amount of work from the

queue itself. There are various versions of this mechanism but the simplest of them provides that, according to some strategy, a negative customer can delete or kill an ordinary-positive customer.
Some extensions of this model include the removal of a random batch of customers due to the arrival of a negative customers. This corresponds to removing all or some random amount of work from the queue which will may not always represent an integer number of normal customers. The definition of negative customers and triggers arises with the need to model server crashes or traffic jams in which some users have to be removed from the system or transferred to another queue.

2.6.2 Definition

G-Networks are networks of n interconnected queues and their main characteristics are:

- all queues in this network (called *G-Queues*), have their own server which ensures its service with a rate of μ_i
- external arrivals, of positive and negative customers or of triggers and resets, have a rate λ_i
- when a positive customer (also called normal customer or simply customer) is serviced, it can have multiple choices of behaviour:
  - go to another queue remaining a positive customer;
  - change its nature and become a negative customer, a trigger or a reset;
  - exit the system.

  The probabilities of these choices are represented usually as p^+_{ij}, p^-_{ij} and d_i, respectively.
- Arrivals of positive customers in a queue, increase the length of the queue by the same amount.
- Arrivals of negative customers in a queue, can reduce the length of the queue by some fixed or random number of customers (if the queue is non-empty and there is at least one positive customer).
- Arrival of triggers in a queue, moves some fixed or random number of positive customers to other queues with some probability.
- Arrivals of resets in a queue, reset the queue and set it in its steady-state only if the queue is empty at the arrival time.
- each negative customer, trigger and reset, will disappear immediately after completion of their actions. For this reason, they are considered *control signals* of the network.

Positive Customers

They are the first type of customers, also known as *regular customers* or *normal customers*. A server treats those customers in the usual way. The dynamic behaviour of the considered network is determined by service and routing disciplines that positive customers follow.

Negative Customers

This kind of customers, opposite to the positive ones, they induce (or kill) a positive customer to immediately leave the node if there is at least one customer in the queue, for this reason they have the effect of a signal

The modelling of neural networks was the first motivation of the introduction of this queueing network with positive and negative customers. A neuron is represented by a node within this context and excitation and inhibition signals are represented by positive and negative customers routing in the network. Those signals increase or reduce the neuron potential in which they arrive by one unit.
G-Networks provide a basis that unifies queueing and neural networks and are the result of an extension of the original network of Gelenbe. Several extensions were made since the introduction of G-Networks like triggered movement, batch service networks, multiple classes networks, state-dependent service disciplines, disasters and tandem networks. G-Networks are a versatile class of networks in which the network behaviour is affected in many ways by the arrivals of negative customers. Some of the different possibilities that have been introduced in this topic are:

- *Individual removal*: the event in which a positive customer is cancelled by a negative customer arrival at the queue. It hasn't effect in case the queue is empty.
- *Batch removal*: the event in which a batch of customers are forced to leave the network due to a negative arrival.
- *Disaster*: the event in which a catastrophe in the node is the effect of a negative arrival at the node. This means that all customers are forced to leave the queue immediately and automatically.
- *Triggered movement*: the event in which the instantaneous forcing, of a customer movement to some other node or a batch of them to leave the network, is triggered by a negative arrival which acts as a signal of that trigger.
- *Random work removal*: the event in which the arrival of a negative customer removes instantaneously a random amount of work from the system.

2.6.3 Basic G-Networks

The first G-Networks introduced by Gelenbe, (i.e., without any extension) are those with only general positive and negative customers. Considering a Markovian network with n servers (also called nodes), the arrival of customers agrees with an independent Poisson streams of rates, Λ_i for positive customers and λ_i for negative ones. Customers are serviced in a server i with times that are exponentially distributed with rate $r(i)$.
If the queue is non-empty when negative customers arrive, its queue length is reduced by one unit. That cancelled unit represents a positive customer removal.
When a server completes service of a customer, it can behave in three ways when leaving the queue i and joins queue j:

- A cusomers remain a positive customer with probability $p^+(i,j)$
- A customer becomes a negative customer with probability $p^-(i,j)$
- A customer leaves the network with probability $d(i)$

In this way, the traffic equations are the following:

$$\lambda^+(i) = \sum_j q_j r(j) p^+(i,j) + \Lambda(i)$$
$$\lambda^-(i) = \sum_j q_j r(j) p^-(i,j) + \lambda(i)$$

where $q_i = \dfrac{\lambda^+(i)}{r(i) + \lambda^-(i)}$ for $i = 1, ..., n$.
These equations are non-linear and this is their main feature. The interesting thing is that if there exists $\{\lambda^+(i), \lambda^-(i)\}$ unique and non-negative solution such that $q_i < 1 \quad \forall i > 0$, then the stationary probability distribution will have the following product-form:

$$p(k) = \prod_{i=1}^{n} (1 - q_i) q_i^{k_i}$$

where $k = (k_1, ..., k_n)$ represent the vector of lengths of the queue.

2.6.4 Stability

Even if a basic G-Network has its product-form solution, it has also non-linear traffic equations. This property distinguishes it from other classical queueing networks. The non-trivial analysis of the stability of the network and of the existence of a solution to its traffic equations is the consequence of this. The stability involves the existence of the stationary probability distribution. Thus conditions for general stability in a multiple class G-Network (a generalization of the "pure" network) are developed. This introduced method is quite general and it can be used to analyse also extensions of G-Networks like triggered movement, batch service, G-Networks with different service disciplines [55, 48, 51, 50, 52].
The main point is to prove that solutions to the traffic equations exist, subsequently the uniqueness of solution is easy to prove. This because it involves the stationary solution of a system of Chapman-Kolmogorov equations.
Moreover, **if a positive solution** $p(k)$ **exists then it is unique**.
Considering the following definition of vectors:

$$\Lambda = [\Lambda(i)], \quad \lambda = [\lambda(i)], \quad \lambda^+ = [\lambda^+(i)], \quad \lambda^- = [\lambda^-(i)].$$

And taking into consideration also:

- the matrices, P^+ and P^-, of elements $p^+(i,j)$ and $p^-(i,j)$ respectively;
- the diagonal matrix F of the element which represents the fraction of queue entering as positive customers, who survive long enough to be served by the server and leave the queue unkilled.

We can write the traffic equations in the following way:

$$\lambda^+(I - FP^+) = \Lambda, \qquad \lambda^- = \lambda^+ F P^- + \lambda^+.$$

Now it can be proven that there always exists the solution $\{\lambda^+(i), \lambda^-(i)\}$ for $i = 1, ..., n$ [48].

2.6.4.1 Important Property

When dealing with G-Networks, the following important property ensures that at equilibrium it doesn't matter from which state of the queue we have started, we will always get the same result we search for. Obviously assuming that conditions for the state equilibrium are satisfied (e.g., it cannot be that positive customers arrival rate is much higher than the sum of the arrival rate of negative customers and servicing rate).
The property is this:

When the single node is at equilibrium,
the streams of future arrival of positive and negative customers,
the past departure processes of positive and negative customers
and the current state of the network are **independent**

2.6.5 G-Networks Extensions

In the last years, a lot of different extensions and possibilities have been studied in this topic of G-Networks. The current literature was made in a short period of time and now I will present the main extensions that were made for G-Networks. This should discriminate the various kinds of G-Networks and what characterizes each of them.

2.6.5.1 G-Networks with Triggered Customer Movement

For what concerns queueing networks in classical literature, regular customers are assumed to move into another queue or leave the system after they have been serviced (namely after some amount of service time). They also usually follow a Markovian routing mechanism.
This is the standard idea of customer transfer and this only occurs after a service completion, however it's possible also that in queueing networks, there are external event occurrences which force and route a customer to leave its queue.
Gelenbe himself introduced this new idea: *G-Networks with positive customers and signals*.
A Poisson process of rate $\lambda(i)$ leads the arrivals of signals from outside the queue.
If the queue is a non-empty one, signals can have two different behaviours (and consequently two kind of consequences in it):

- arrival of a signal implies the movement of a customer with probability $q(i,j)$ from its queue i to another different queue j;
- arrival of a signal trigger a customer (or a batch of them) to leave the network, from queue i with probability $D(i) = 1 - \sum_j q(i,j)$.

With the combination of this description and the routing discipline, after some service is completed, we can comprehend that a signal could be defined by the various movements of regular customers or it could also be exogenous. In both cases, an additional facility, represented by signals, is added to the network for allowing other movements of positive customers through the system.
There are other extensions of this type where signals have a similar behaviour to triggers. Those are networks with both positive customers and signal emissions. The emission of signals for the moving of a batch of customers and the concept of triggers are quite similar to each other.

Other extensions, for instance, involve also the use of signals within the system for moving a customer but considering them like a secondary service process. Those processes have parallel characteristics with respect to regular servicing events. This leads to the superposition of mechanisms of two services and so these signals are no more an extra facility for the movement of regular customers.
Some multiple class G-Networks were also developed, with random triggering times. In these systems after the arrival of a signal in the queue, there is some random delay before the trigger of a customer movement. The class of the leaving customer, the queue in which it is and the source of the signal define the random variable which represents that amount of time.
One last example is the incorporation, into probabilities of routing, of some degree of history dependence. This led to two different models of multiple class of G-Networks:

- when a service is completed or when a negative customer arrives, the amount of service that has been received defines the various routing probabilities.
- after the completion of a customer servicing time or after its obliged movement due to a signal, the cause of its movement (namely for signal arrival or end of its servicing) and the number of times in which signals interrupted its service define the various routing probabilities.

2.6.5.2 G-Networks with Batch Removal

In G-Networks with batch removal, customers can exit the network in batch mode. The variations assume different batch distributions and also different movements through the system of signals and batches.
The first extension was made with the possibility of batch service when there is a negative arrival. This kind of model keeps assumptions of input and behaviour when a regular service ends. On the other hand, when there is a negative arrival on a non-empty queue, the effects are the same as of the arrival of a signal. The consequences are two:

- after that signal arrives, a movement of a normal customer, with probability $q(i, j)$ from queue i to queue j, is triggered;
- with the signal arrival, a service of a batch of normal customers is completed.

The batch size has a random distribution B_i and it only depends on the queue i in which the signal came. In case of a lesser number k_i of customers in the node with respect to B_i, all and only the available regular customers are removed.
There is also the extension which assumes that signals (or negative arrivals) are secondary servicing processes. In this way both movements, caused by a service completion of normal customers or by the arrival of signals, are the same but they have their own specific parameters of movement triggering.
It can also be the case that a model has two kinds of different events in a queue i after a service completion:

- type I: with probability p_i, the routing probabilities agree with a general G-Network.
- type II: with probability $1 - p_i$, a batch of size B_i leaves the network

Another batch removal system has also a symmetric behaviour between service completion and negative arrivals time but implies the following operating rules when there is a completion of service in the queue i:

- Customers in a queue can merge together in incomplete or partial batch;
- if a full batch is moved into a queue with a partial one, the latter is automatically removed.

The routing possibilities are the following:

- the full batch joins the queue as a regular and single customer with probability $p^+(i,j)$
- the full batch joins the queue as a signal with probability $p^-(i,j)$
- the full batch leaves the system with probability $d(i)$

There can be a further implementation in which both origin and destination queues can define the formation of negative customer batches caused by a triggering movement. For example, the completion of a service in a queue i can force the movement of a departing customer to another queue j and in the meantime also the simultaneous arrival in the same queue j of a batch of n negative customers with probability $d_{ij}(n)$. These creations of negative batches can also be triggered by the external arrival to node i of a positive customer (with size n and probability $d_{0i}(n)$).

2.6.5.3 Multiple Class G-Networks

Another family of alternative is represented by the case of multiple classes of both positive and negative customers. In these extensions, either type of customers can belong to C different classes. An independent Poisson stream of arrival of positive (with rate Λ_{ic}) and negative (with rate λ_{ic}) customers in queue i characterize each class c. At queue i, c-type positive customers are served with exponential servicing time distribution r_{ic}. In the case of a normal G-Network $\lambda_{ic} = \lambda_i, \forall c$.
In those multi-class G-Networks, the state of system is now defined by:

- vector $k = (k_1, ..., k_n)$
- each component $k_i = (k_{i1}, ..., k_{iC}) \quad \forall i = 1, ..., n$
- each k_{ic} represents the number of customers of class c in queue i

Negative customers effects can be described in various ways:

- the negative arrivals of a c-class in the queue i affect only positive customers of the same type (i.e., the same class c). Thus, if the number of c positive customers is greater than zero ($k_{ic} > 0$) then the length of the queue is reduced by one unit.
- the negative arrivals trigger a removal of a customer of a random class. Thus, if in queue i a negative customer arrives and it is a non-empty queue ($k_i > 0$), then a positive customer of class c is removed with probability $\frac{k_{ic}}{k_i}$.
- the most intricate of the three removal policies: the negative arrivals to empty queues don't have any effect and simply leave the system. The negative arrivals to non-empty queues select a regular customer to be removed, according to the service discipline of the current queue. Then, the negative customer tries to kill the selected regular one and succeeds with probability $K_{i,m,n}$, where i is the queue, m is the class of the negative customer and n is the class of the positive customer.

There are further studies in which several service disciplines have been considered in these G-Networks extensions like:

- first-come-first-served (FIFO or FCFS)
- processor sharing
- last-come-first-served (LCFS or LIFO) with preemptive resume priority
- effort depending on the state of the system
- symmetric queues
- arbitrary service times

Moreover, there is the introduction of multiple classes also for triggers.
Also the routing probabilities can be changed in some models depending on the service times or on the number of service interruptions. Usually those G-Networks have n nodes, C classes of positive customers and only a unique type of negative ones. In this way, node i and class c determine the arrival rates Λ_{ic} whereas only the node i determines the exogenous negative arrival rate λ_i.
We have to remember that different G-Networks usually require various service requirements. Generally speaking, different servicing time distributions belong to different types of customers but in some cases, to soothe any other condition we assume homogeneous service requirements.
For what concerns the deletion choice of customers, it could be:

- random
- depend on the position of the customer in the queue
- the class of the customer
- the service effort offered

All these multiple class G-Networks can be described also as a particular case of a general state-dependent description. Those more general systems have state-dependent intensities and their signals can be either a trigger or a batch of negative customers. Moreover, two non-negative functions, ϕ and ψ (defined on $\mathbb{Z}^n$), describe the arrival of signals and positive customers.

2.6.5.4 G-Networks with Disasters

In some extensions, G-Networks can also be composed by the presence of a flow of disasters. Contrary to previous G-Networks, in this case there is the possibility of a disaster. These disasters represent an extreme case of customer exodus. In fact, all the customers present in the queue, which is affected by the disaster, are automatically removed from the system.
The first studies of a clearing mechanism for single node queues were further developed in more detail with G-Networks with disasters. Virus infections in computer networks and catastrophes in migration processes are some possible applications to those models. All the assumptions of the main (or "pure") G-Network remain valid with one exception for the negative arrivals. In this specific case, a Poisson flow of disasters replace the usual single negative arrivals. These disasters arrive in a queue i with rate $\lambda(i)$ and the rules

for arrivals and routing of disasters are the same as those for negative arrivals. The only difference is on the behaviour, a disaster implies the total destruction of all customers (and indeed of work) in the destination queue.
In this kind of network, the following non-linear system is satisfied by their traffic equations:

$$\lambda^+(i) = \sum_j q_j r(j) p^+(j,i) + \Lambda(i),$$
$$\lambda^-(i) = \sum_j q_j r(j) p^-(j,i) + \lambda(i)$$

where

$$q(i) = \Big\{\lambda^+(i) + \lambda^-(i) + r(i) - (\lambda^+(i)^2 + \lambda^-(i)^2 + r(i)^2 + \\ +2\lambda^+(i)\lambda^-(i) + 2\lambda^-(i)r(i) - 2\lambda^+(i)r(i))^{1/2}\Big\}/2r(i)$$

If either $\lambda^-(i) > 0$ or ($\lambda^-(i) = 0$ and $\lambda^+(i) < r(i)$) are satisfied by the solution of this system for each node i, then the stationary probability distribution will have the product-form:

$$p(k) = \prod_i (1 - q_i) q_i^{k_i}$$

2.6.5.5 Tandem G-Networks

Usually the main analysis in G-Networks includes stability, balance equations and existence of a product-form. In the case of tandem G-Networks, the delay times and the response time distribution are the focus of analysis.
In a tandem of two Markovian G-Queues, the arrivals of customers in the queues are determined by Poisson processes. With rate $\Lambda(i)$ for positive customers and rate $\lambda(i)$ for negatives one (for $i = 1, 2$ where i is the number of the queue). Moreover, regular customers are served with time exponentially distributed with rate $r(i)$. We have to remember that the exponential law has the memoryless property, thus the policies of service and removal, and the queue length distribution are independent.
In this way the limiting probability distribution is the following:

$$p(k_1, k_2) = \prod_{i=1}^{2} (1 - \rho_i) \rho_i^{k_i}$$

where

$$\rho_1 = \frac{\Lambda(1)}{\lambda(1) + r(1)} < 1,$$

$$\rho_2 = \frac{r(1)\rho_1 + \Lambda(2)}{\lambda(2) + r(2)} < 1$$

To analyse the response times distribution (namely end-to-end delays) it is assumed that queueing discipline in the G-Queues pair is FCFS and in case of negative arrival, the removal of customers starts at the end of the queue (RCE). However this computation of the response time distribution also for simple G-Networks is quite complex due to a phenomenon of overtaking. This event is the supervision of a positive customer from its removal: when another regular customer arrives in the queue immediately after the

observed customer, positive customer protects the observed one from the removal due to a negative arrival. In this way the fate of observed customer depends on the arrival of positive customers after it.
The non-independence in each queue of sojourn times, which on the contrary exists in classical networks without negative arrivals, is explained by this phenomenon.
The probability of a positive customer not being deleted, jointly with the distribution of response time, have the following form (in terms of the Laplace transform):

$$W^*(s) = (1-\rho_1)(1-\rho_2) \times \frac{r(1)}{\Lambda(1)} y_1(s) G(\rho_2, 0, y_1(s), s)$$

Considering a Markovian G-Queue with FCFS and RCE discipline, in this formula the root involved in the sojourn time distribution is represented by $y_1(s)$ and functional equation is satisfied by function G. Dealing with the dependence of sojourn times leads to this complex solution, even though the probability that a customer is not removed can be reduced to the following simple product formula:

$$W^*(0) = \frac{r(1)}{\lambda(1) + r(1)} \quad \frac{r(2)}{\lambda(2) + r(2)}$$

If we consider the particular case in which $\Lambda(2) = \lambda(2) = 0$, then, as we can expect, the sojourn times are independent in all queues and moreover distributed like considering each corresponding queue to be isolated.
Finally, in the case of $\Lambda(2) = \lambda(1) = 0$, then customers in the second queue are protected by the customer departures of first queue and consequently there is still dependence on the sojourn times.

2.6.5.6 The Service Mechanism

Now I will briefly describe the different possible choices of the service mechanism and other relating operating rules for G-Networks. This includes also the individual service times. The network design of the whole system is strictly connected with the choice of a service mechanism, this means that we have to strengthen the description of other network components if we relax any assumption on the choice. The final outcome and also the service facilities are greatly influenced by those compensations. In general, $r(i)$ is the rate of exponential service distribution of time in basic G-Networks at queue i but it can be changed when there are multiple classes of customers and it is modified by the type of customers and/or the queue considered.
In a multiple-class G-Network with three distinct types of service centers which have the following service disciplines:

- First-Come-First-Served (FCFS or FIFO)
- Processor Sharing (PS)
- Last-Come-First-Served with preemptive resume priority (LCFS/PR or LIFO/PR)

Each class of positive customers c can obey a different service rate r_{ic}. In this case, $k = (k_1, ..., k_n)$ is the representation of the state of FCFS and LCFS/PR queues and in the queue i, the vector (k_{ij}) represents the state k_i. The length of that vector is the number of positive customers in the node and the class of customers in position j is represented by the j-th element of the vector. On the contrary, with PS queues, the state k_i is itself a vector and the number of positive customers of class c in node i is represented by the

c-th element of the vector. In some cases, G-Networks (with general $n+1$ queues) can label the queue 0 as the "outside" of the system and allow the queues to have a negative queue.
Considering the general state-dependent intensities, we can define each node as a source and a customers queue. The rate of generation of positive customers is the following:

$$\Lambda(i)\frac{\psi(k-e_i)}{\phi(m)}$$

with $k=(k_1,...,k_n)$ is the vector of lengths of the queue, e_0 is a vector of zeros with length n and e_i (with $i>0$) is a n vector of zeros except of a 1 in the i-th position. Moreover, $\psi(\cdot)$ is a non-negative function in $\mathbb{Z}^n$ and $\phi(\cdot)$ is a positive function in $\mathbb{Z}^n$.
There are also the following definitions:
d_{ij}^+ is the probability of the routing to queue j of the emitted customer, with $0\leq j\leq n$ and the subsequent addition of one unit in the length of queue j (except for queue 0),
$d_{ij}^-(t)$ is the probability of transformation of the emitted customer into a signal t and its routing to queue j, with $0\leq j\leq n$ and the negative batch size is denoted by t ($t\geq 0$)
Then the emission embeds the departure process and the choice of probabilities d_{ij}^+, $d_{ij}^-(t)$ determines triggers and individual (or batch) services dependence.
Let's consider now, mechanisms of service when they are state-dependent, having also symmetric queues and general service times. Furthermore, the system with state-dependent service times is a multiple class G-Network. A positive customer in a queue i, requires a servicing time exponentially distributed with rate $r(i)$, so it doesn't depend on the type of class. When k_i customers are in the queue i the rate of total service is: $\Phi_i(k_i)r(i)$.
A customer, in position l with $l=1,...,k_i$ of the queue i, has an effort service that is a $\gamma_i(l,k_i)$ proportion of the total potential of service. After the completion of the service of that customer, other customers in positions $l+1,...,k_i$ shift to positions $l,...,k_i-1$. A positive or signal arrival trigger the reallocation of customers and this reallocation is determined by two auxiliary probabilities $\delta(l,k_i)$ and $\eta(l,k_i)$. Moreover, standard routing probabilities connect individual queues with one another.
Finally, if we relax the assumption of class homogeneity, assuming that service requests of customers of class c in queue i have exponential time with rate r_{ic}, then we have to make an assumption that the discipline of service is symmetric ($\gamma_i(l,k_i)=\delta(l,k_i)=\eta(l,k_i)$).
There can be also other models with different assumptions of servicing efforts and the analysis could involve and arbitrary service time distribution $F_{ic}(x)$. In conclusion, even if it depends on the service time distributions, the stationary distribution has also its own product-form.

2.7 Applications

There are a lot of different fields of applications, in which complex systems can be modelled by the class provided by G-Networks, such as:

- Computer Networks
- Neural Networks
- Migration Processes
- Telecommunication Systems
- Production Systems

- Maintenance

From the original "pure" G-Networks in which inhibitor signals of neural networks are represented by the negative customers, there has been a wide and huge extension of that first model. All those applications now can cover a lot of different fields.

Chapter 3

Compositional Modelling

3.1 Introduction

This is the last introductory chapter, in which we describe compositional modelling. We illustrate Markovian process algebras and their main results for the product-forms. In the first part we focus on continuous time models with the Performance Evaluation Process Algebra (PEPA) and with Stochastic Automata (SA). After a brief description of discrete time process algebras, we recall the Probabilistic Input/Output Automata (PIOA) and its formal description. In the last part of this chapter we talk about product-forms, and special attention is devoted to the presentation of RCAT theorem and the notion of quasi reversibility which are the basis from which our work starts. We also present the main theorem on product-form QNs (i.e., BCMP theorem) and we illustrate the properties on the queueing model that imply the BCMP product-form. These properties can be expressed in terms of a characterization of the scheduling discipline or in terms of properties of the underlying CTMC (e.g., Local Balance Property or $M \implies M$ property).
The class of product-forms is wide and includes different kinds of systems. Even if the Reversed Compound Agent Theorem (RCAT) [61] and Extended Reversed Compound Agent Theorem (ERCAT) [63] are very general results for the analysis of product-form stochastic models, they have some limits and do not capture all product-forms models, as we will see in the last part of this chapter.

3.2 Performance Evaluation Process Algebra (PEPA)

3.2.1 General View

The subject to capture and analyse the dynamic behaviour of a network, a communication system or a queue is the performance modelling.
Unfortunately it is difficult to study and analyse many modern systems, due to their size and complexity, resulting in very huge and complex models. To solve this problem, the drive to take a compositional approach arises. This approach decomposes the whole system into smaller and easier subsystems. They are indeed less complicate and consequently more easy to model. One of the main compositional approaches to performance modelling is based on PEPA (Performance Evaluation Process Algebra) [70].
PEPA is a suitably enhanced process algebra and its language has a pronounced compositional nature. This can provide a lot of benefits for the solutions of models and in the same way for model construction.

PEPA provides an operational semantics and uses it to create an underlying Markov process. It may be used also as a paradigm for specifying Markov models. This can be done for any PEPA model and the method is well explained and demonstrated in [70].
To deal with the problems of large performance models, some techniques of model simplification and state space aggregation have been proposed. They were developed as notions of equivalence between entities of models and have an intrinsic interest from both the process algebra perspective and usefulness in the context of performance modelling. The basis of these model transformation techniques is formed by equational laws. These laws are generated and ensured by a strong structural equivalence, the isomorphism. Using also the abstraction capacity of the PEPA modelling, this equivalence leads to a technique of model simplification. This, obviously, providing that certain conditions are satisfied.
These tools may be used to replace one model component by another one which must have the same apparent behaviour.
Moreover, limiting only to timed and probabilistic behaviours of systems in a process algebra, may result to be less suitable for performance modelling. For this reason, PEPA, as a stochastic process algebra, is effective in modelling complex and real systems [70]. This process algebra tries also to identify problems of performance evaluation in order to offer a systematic method for modelling also complex systems.
To facilitate the analysis of the whole system, separate components and aspects of it may be considered individually. They can be also subsequently considered in a more abstract form, as their interactions are developed. Furthermore, the methods of model simplification of PEPA can avoid the generation of the complete state space of the underlying stochastic process.
Finally all those methodologies have been formally defined with its operational semantic, in this way they could lead to automation or machine-assistance for model simplification.

3.2.2 Main Features

One of the main targets of a process algebra able to make performance evaluation is to capture as many features of a normal process algebra as possible while also having characteristics that allow specification of stochastic processes. In this context, the performance evaluation part can be seen as an extension, taking also the normal features of normal process algebras as a basis to be used as a design formalism and thanks to annotations of the design we can develop the performance model.
The following features are considered essential for this process algebra:

- Parsimony: elements of the languages must be few in numbers and also simple. With this parsimony the reasoning on the language should be easy and ensure flexibility in the modelling phase. The basic elements of this language are components and activities, corresponding in the underlying stochastic model to states and transitions.

- Formal Definition: in the language there is a structured operational semantics, this provides formal interpretation for each expression. These rules are the basis of the notions of equivalence, giving a formal way to compare and manipulate both models and components.

- Compositionality: the combinator of cooperation is the basis of composition in PEPA. Complementary to it, there are methods of model simplification and aggre-

gation. In this way a part of a model can be simplified in isolation and replaced by a simplified component without compromising the integrity of the whole model.

Quantification of time and uncertainty are main attributes needed for performance evaluation. They are present in PEPA but missing in other process algebras such as CCS. In other algebras time of actions is usually implicit and the models are non-deterministic but it is important to quantify timing behaviour and uncertainty for extracting performance measures from performance models.
PEPA associates random variables (representing duration) to all activities, to achieve this. In this way, timing behaviour of the system is represented by delays of each activity in the model. Furthermore, also temporal uncertainty (regard actions duration) is captured because the duration is a random variable.
Succession uncertainty (about next events) is obtained because probabilistic choices replace non-deterministic ones. A race condition between the enabled activities determines probabilities of branchings.
Introducing random variables for all system activities can be considered as annotations of the pure process algebra model.

3.2.3 Language - Informal Description

With PEPA you can describe a system via interactions of its components and actions those elements can engage in. Parts and behaviour of the system will correspond to some components. A queue could be represented by an arrival component and a service component, both of them will interact to form the queue behaviour.
Moreover, each component can be atomic or can be composed itself by other components and it is assumed that there is a countable number of possible components, each of them has a behaviour defined by its possible activities. Contrarily to other process algebras, in PEPA a random variable representing duration, with exponential distributions, is associated to each activity. Every activity has a type (called *action type*) from a countable set A, of all possible types. In case that different activities have the same type, this means that they are different instances of the same action in the system.
The action type τ, called also unknown type, represents an action (or a sequence of them) that is unknown or not important to identify. This kind is considered private to the component in which it occurs and it is not instantaneous, like any other type of action. Nonetheless, different instances of τ don't necessarily represent the same action within the modelled system and they could be represented by their real number parameter of duration.
Generally speaking, this parameter is called the activity rate and must be greater or equal than 0 or $\top$ which means that it unspecified.
There will be the following conventions about the names of various elements:

- Components are denoted by names starting with large roman letters (P,Q,R,P_i,...).
- Activities are denoted by single small roman letters from the alphabet beginning (a, b, c,...).
- Action types are denoted by small greek letters (α,β,γ,...), or by names starting with a small roman letter ($task$, $service$, $check_i$,...).
- Activity rates are denoted by single roman letters from the alphabet ending (r,s,t,s_j,...). Usually the greek letters μ (for the service rate) and λ (for the arrival rate) are also used.

- Subsets of A are typically denoted by L, K, and M.

In this way, each activity is defined by pairs such as $(\alpha; r)$ where $\alpha \in A$ is the action type and r is the activity rate.
From this, we can say that there exists a set of activities, $Act \in A \times \mathbb{R}^+$, where $\mathbb{R}^+$ is the set of real positive numbers and the symbol $\top$.
There is also some other terminology to introduce:

- System behaves as P: when component P determines its behaviour
- $\mathcal{A}(P)$: current action types of P. Component P can next perform only these action types.
- $\mathcal{A}ct(P)$: multiset of current activities of P. Component P can next perform only these activities.
- Delay period of an enabled activity $a = (\alpha, r)$: period of time given by its associated distribution function that is the probability that this activity happens within a period of time t, with $F_a(t) = 1 - e^{rt}$.
- P': is the component which describes the behaviour of the system when P completes α for some $\alpha \in \mathcal{A}ct(P)$. P' does not have to be different from P.
- $P \xrightarrow{\alpha} P'$ or $P \xrightarrow{(\alpha,r)} P'$: completion of activity α and the subsequent behaviour of the system as P'.

The distinction between action types and activities is the dynamic behaviour of a component which depends on the number of instances of each enabled activity. Moreover, $\mathcal{A}(P)$ is a set and $\mathcal{A}ct(P)$ and is a multiset (unless stated otherwise).
The delay period is like a timer set by an activity when it becomes enabled and the rate of the activity determines its time. Furthermore, each enabled activity has its own associated timer and the first which finishes implies that its corresponding activity takes place (an external observer could observe the event of an activity of that type). That activity is said to be succeeded or completed and others are considered preempted, or aborted.

3.2.4 Language - Syntax

The primitives of PEPA language are: components and activities. Remember that activities characterize the behaviour of a component and also PEPA has a small set of combinators. Those combinators allow us to construct expressions and terms which can define the behaviour of components through interaction between them and activities they can perform. In this way, the behaviour of components can be influenced also by the environment in which they are placed.
In PEPA, the syntax of terms is defined as follows:

$$P ::= (\alpha, r).P \mid P + Q \mid P \underset{L}{\bowtie} Q \mid P/L \mid A$$

Now we will give a brief description of names and interpretations of those constructions.

Prefix: $(\alpha, r).P$
The basis of the construction of component behaviour is the mechanism of Prefix. In this

case, the component $(\alpha, r).P$ can perform the activity (α, r), which has action type α and a duration exponentially distributed with parameter r (mean $\frac{1}{r}$). For completing the activity, the time is t, drawn from the distribution. After its completion, the behaviour of component will be as component P. If a component reaches the behaviour of $(\alpha, r).P$ at some time t', it will complete the action (α, r) in $t' + t$ time. In this way it will become P, enabling all the activities in $\mathcal{A}ct(P)$. If we consider $a = (\alpha, r)$ then we can write the component $(\alpha, r).P$ as $a.P$.
We have to recall also that we assume there is always an underlying implicit resource, which helps the component activities. They are not modelled explicitly but their utilization by the components are represented by the time elapsed before the completion of an activity. Those resources could be time processor, cycles of CPU, some I/O devices or bandwidth of a communication channel; this can depend on which system and at which level it is modelled.

Choice: $P + Q$
A system which can behave both as component P or as component Q, can be represented by the component $P + Q$. This component enables all the current enabled activities of P and all the current enabled activities of Q at the same time.
To be more formal: $\mathcal{A}ct(P + Q) = \mathcal{A}ct(P) \uplus \mathcal{A}ct(Q)$ (where $\uplus$ denotes the multiset of the union) and whatever enabled activity will be completed, it must belong to $\mathcal{A}ct(P)$ or $\mathcal{A}ct(Q)$. It must be in this way, also in the case that P and Q have the same enabled activity since different instances of the same activity are distinguished in PEPA. Doing so, the future behaviour will be of P or of Q and it will be distinguished by the first activity to complete. This activity will also discard the other component of the choice.
We have an important thing to remark, **the probability, that both P and Q complete an activity at the same time, is zero**, and this is ensured by the continuous nature of the probability distributions. If we consider that after P completed the activity, it behaves as P' component and similarly Q and Q', then the system will subsequently behave as P' or Q' since $P + Q$ had completed an activity.
Finally, we have to consider that there is always the underlying assumption that either P and Q compete for the same implicit resource.
In this way, the competition between components is represented by the choice combinator.

Cooperation: $P \underset{L}{\bowtie} Q$
The set L is called the *cooperation set*, and the interaction between the P and Q components are determined by it. If we consider that $K \neq L$, then almost certainly component $P \underset{L}{\bowtie} Q$ will behave quite differently from the behaviour of $P \underset{K}{\bowtie} Q$. This makes the cooperation combinator, an indexed family of combinators, one different combinator for each possible set of action types $L \subseteq A$. The action types, on which the components must cooperate or synchronise, are indeed defined by the cooperation set.
Contrary to the choice operator, we assume that in a cooperation, each component has its own implicit resource. Moreover, they independently proceed with any other activities whose types are not in the cooperation set L but those whose action type is in the set L must involve simultaneously both components (and consequently both resources) in an activity of that type. Any cooperation set can't contain any unknown action type τ.
The activities which have types which do not occur in L, are called individual activities of the components and they will proceed unaffected.
On the other hand, shared activities, whose types are in the set L, will only be enabled in

$P \underset{L}{\bowtie} Q$, namely when they are enabled in both P and Q. Doing so, one component, which wait for the other component participation, can become blocked and represent situations in the system when, to achieve an action, the components have the need to work together. Generally speaking, each component has to complete some work. This work corresponds to their own representation of the action and individual activities of the individual components P and Q are replaced by a new shared activity, formed by the cooperation $P \underset{L}{\bowtie} Q$. The shared rate of this new activity reflects the rate of the slower participant even if the activity will have the same action type as the two contributing ones. Thus this rate (namely the expected duration) of a shared activity, will be greater than or equal to the rates (namely the expected durations) of the corresponding activities in the cooperating components.
In the case of an unspecified rate of an activity in a component, the component is called *passive* with respect to that action type. In other words, the component will not contribute to the work involved, even though its cooperation can be required to achieve the completion of an activity of that type. One example can be the role of a channel in a communication system: if we want to transfer a message, the cooperation of the channel is essential, but the transfer don't need any work (i.e., consumption of implicit resource) of the channel itself.
Finally, if the set L is empty, $\underset{L}{\bowtie}$ has the same effect of a parallel composition. This allows the components to proceed concurrently without any kind of interaction between them and this kind of situation will occur quite often, in particular when systems have repeated components. For this reason to represent $P \underset{\varnothing}{\bowtie} Q$, there is the introduction of the more concise notation $P||Q$. The combinator $||$ is called parallel combinator.
We have to notice that also with this syntactic convenience inclusion, there is no expressiveness addition to the language.

Hiding: P/L
In the component P/L, all activities, whose types are in the set L, are hidden, this means that their completion cannot be witnessed and this is the only difference between P/L and P behaviour (i.e. an observer can only witness a hidden activity by its delay). Those hidden activities will appear as the unknown type τ and they can represent an internal delay of the component.
The activities that a component can engage individually are not affected by hiding but it affects the possibility to fully witness externally these activities. If a process completes an activity, an external observer can see the type of that completed activity and can also be aware of the length of time from the previous activity completion, i.e., the delay of time in which the activity took place.
Furthermore, those hidden activities cannot be within a cooperation set L with any other component and their action types are no longer accessible outside to both another component or external observers. Anyhow, activities are not affected in their durations even if they are hidden.

Constant: $A \stackrel{def}{=} P$
Assuming that the set of constants is countable, the meaning of constants component is defined by equations such as $A \stackrel{def}{=} P$, in this way the behaviour of component P is given to the constant A. Doing so, names can be assigned also to components (behaviours).
Let's suppose that E is a component expression containing also a variable X, the expression $E\{P/X\}$ will then indicate the component made by the replacement of P in every

occurrence of X in E. Generally speaking, if we consider an indexed set of variables, then $E\{\tilde{P}/\tilde{X}\}$ is the replacement of an indexed set of variables $\tilde{X}$ by an indexed set of components $\tilde{P}$.

In PEPA the precedence of combinators is defined and this provides a default interpretation of any kind of expression.
The precedence, from the highest to the lowest, is the following:

1. Hiding
2. Prefix
3. Cooperation
4. Choice

Moreover, forcing alternative parsings or simplifications, to clarify meaning, can be done using brackets.
They can also be used to clarify the meaning of a combination of components. For example, $P \underset{L}{\bowtie} Q \underset{K}{\bowtie} R$ has an unclear scope of the cooperation sets L and K. There are two alternatives:

- $(P \underset{L}{\bowtie} Q) \underset{K}{\bowtie} R$: in this case R, for each action type in $L \setminus K$, can proceed independently but it has to cooperate with $P \underset{L}{\bowtie} Q$ for any action types in K
- $P \underset{L}{\bowtie} (Q \underset{K}{\bowtie} R)$: here $Q \underset{K}{\bowtie} R$ has to cooperate with P to perform any action types in L but P is free to do independently each action types in $K \setminus L$.

In this way the intended scope of the cooperation set can be delimited by the use of brackets and, in case of missing brackets, it is assumed the left association of cooperation combinator. Considering this and using differing cooperation sets in the cooperation between several different components, we can build layers and levels. Each one of these layers uses a cooperation combination of just two processes, which can be formed in turn by cooperation between lower-level components.
One example can be the following:

$$\left((P_1 \underset{L}{\bowtie} P_2) \underset{K}{\bowtie} P_3\right) \underset{M}{\bowtie} \left(P_4 \underset{N}{\bowtie} P_5\right)$$

This component can be seen as $Q_1 \underset{M}{\bowtie} Q_2$ at the top level. Considering $\equiv$ the syntactic equivalence, we have that at lower level, $Q_1 \equiv Q_3 \underset{K}{\bowtie} P_3$ and $Q_2 \equiv P_4 \underset{N}{\bowtie} P_5$ and in the lowest level $Q_3 \equiv P_1 \underset{L}{\bowtie} P_2$.
In the lowest level, components which do not contain any cooperation are called *atomic components*, instead in the top level these are called *top-level components*.

Passive Activities: $\top$
In some cases, the components cooperation is not equal and this can represent that one of them is passive with respect to an action type. This means that each enable activity of that type in the component, has an unspecified activity rate. Those passive activities must be shared with other components which will determine the real rate of this shared activity. A model is called incomplete, if at least one of its passive components is not shared with

others or a cooperation set restricted it.
There is also the case in which more than one passive activity type is simultaneously enabled within a component and we must assign a weight to all unspecified activity rates. Those weights have to be natural numbers and they can be used to determine the relative probabilities of the possible outcomes (regarding the activities of that action type).
If we consider this following component:

$$(\alpha, w_1\top).P \ + \ (\alpha, w_2\top).Q$$

It is passive with respect to the action type α and when that action will be completed, subsequently the component may behave as P with probability $w_1/(w_1 + w_2)$ or may behave as Q with probability $w_2/(w_1 + w_2)$.
Furthermore, we can assume that $(\alpha, \top)$ is an abbreviation for $(\alpha, 1\top)$ and that if there is no weights assignation, then the various instances have all equal probabilities to occur.
From this, comparisons and manipulations of unspecified activity rates are defined by the following inequalities and equations:

$$\begin{aligned}
r &< w\top && \forall r \in \mathbb{R}^+ \quad \text{and} \quad \forall w \in \mathbb{N} \\
w_1\top &< w_2\top && \text{if } \ w_1 < w_2 \quad \forall w_1, w_2 \in \mathbb{N} \\
w_1\top + w_2\top &= (w_1 + w_2)\top && \forall w_1, w_2 \in \mathbb{N} \\
\frac{w_1\top}{w_2\top} &= \frac{w_1}{w_2} && \forall w_1, w_2 \in \mathbb{N}
\end{aligned}$$

Apparent Rate: $r_\alpha(P)$ Usually it could be convenient that a single action of the system is represented by more than only one activity in the model at a time. Nevertheless, the apparent rate of those activities type will always be the same to an external observer of the system or of the model. This is because the race condition of the model assures that the α-rate at which an α activity is completed is the sum of the rates of all the enabled activities of type α.
For example, there exist systems with the multiple capacity of performing an action, like a queue with multiple servers and n waiting customers with obviously $n > 1$. Now let's consider the apparent rate of *service* action of a PEPA component which enables only one type of *service* activity but with a rate n-times the actual *service* rate of the first presented example. The apparent rate of two service activities would be the same.
From that we can comprehend that the apparent rate, at which action types happen, is very important in the comparison of models with systems and between models themselves.
The formal definition for the **Apparent Rate** is:
"The apparent rate of action of type α in a component P is denoted $r_\alpha(P)$ and it is the sum of the rates of all activities of type α in $\mathcal{A}ct(P)$".
And the formal rules are the following:

$$
\begin{array}{lrcl}
1. & r_\alpha((\beta, r).P) & = & \begin{cases} r & \text{if } \beta = \alpha \\ 0 & \text{if } \beta \neq \alpha \end{cases} \\
2. & r_\alpha(P + Q) & = & r_\alpha(P) + r_\alpha(Q) \\
3. & r_\alpha(P/L) & = & \begin{cases} r_\alpha(P) & \text{if } \alpha \notin L \\ 0 & \text{if } \alpha \in L \end{cases} \\
4. & r_\alpha(P \underset{L}{\bowtie} Q) & = & \begin{cases} \min(r_\alpha(P), r_\alpha(Q)) & \text{if } \alpha \notin L \\ r_\alpha(P) + r_\alpha(Q) & \text{if } \alpha \in L \end{cases}
\end{array}
$$

Remember that an apparent rate can also be unspecified, for example if P is defined like this:

$$P \stackrel{def}{=} (\alpha, w_1\top).P_1 + (\alpha, w_2\top).P_2$$

Then by the previous definitions: $r_\alpha(P)$ will be equal to the sum $r_\alpha(P_1) + r_\alpha(P_2)$, so we can sum $w_1\top + w_2\top$ and we know that this is equal to $(w_1 + w_2)\top$. In conclusion $r_\alpha(P) = (w_1 + w_2)\top$.

Current Action Types: $\mathcal{A}(P)$
The set of action types which are enabled by a component P is called $\mathcal{A}(P)$. This set contains all possible action types which can be seen in the next completion of an activity, when the system behaves as component P.
For any PEPA component, this set (**Set of Current Action Types**) can be constructed by the following definitions:

$$
\begin{array}{lrcl}
1. & \mathcal{A}((\alpha, r).P) & = & \{\alpha\} \\
2. & \mathcal{A}(P + Q) & = & \mathcal{A}(P) + \mathcal{A}(Q) \\
3. & \mathcal{A}(P/L) & = & \begin{cases} \mathcal{A}(P) & \text{if } \mathcal{A}(P) \cap L = \varnothing \\ (\mathcal{A}(P) \backslash L) \cup \{\tau\} & \text{if } \mathcal{A}(P) \cap L \neq \varnothing \end{cases} \\
4. & \mathcal{A}(P \underset{L}{\bowtie} Q) & = & (\mathcal{A}(P) \backslash L) \cup (\mathcal{A}(Q) \backslash L) \cup (\mathcal{A}(P) \cap \mathcal{A}(Q) \cap L)
\end{array}
$$

Current Activities: $\mathcal{A}ct(P)$
The multiset of current activities of P is called $\mathcal{A}ct(P)$. It will also take a relevant part in the analysis of a component P. These are the enabled activities when the system behaves as component P.
Adopting the following abbreviations:

$$
\begin{array}{l}
\mathcal{A}ct_{\backslash L}(P) = \{|(\beta, r) \in \mathcal{A}ct(P) | \beta \notin L|\} \\
\mathcal{A}ct_{\cap L}(P) = \{|(\beta, r) \in \mathcal{A}ct(P) | \beta \in L|\}
\end{array}
$$

Then this multiset (**Activity Multiset**) can be constructed by the following definitions:

1. $\mathcal{A}ct((\alpha, r).P) = \{|(\alpha, r)|\}$

2. $\mathcal{A}ct(P + Q) = \mathcal{A}ct(P) \uplus + \mathcal{A}ct(Q)$

3. $\mathcal{A}ct(P/L) = \mathcal{A}ct_{\backslash L}(P) \uplus \{|(\tau, r) \mid (\alpha, r) \in \mathcal{A}ct_{\cap L}(P)|\}$

4. $\mathcal{A}ct(P \underset{L}{\bowtie} Q) = \mathcal{A}ct_{\backslash L}(P) \uplus \mathcal{A}ct_{\backslash L}(Q) \uplus$
$\uplus\{|(\alpha, r) \mid \alpha \in L, \exists(\alpha, r_1) \in \mathcal{A}ct_{\cap L}(P), \exists(\alpha, r_2) \in \mathcal{A}ct_{\cap L}(Q),$
and $r = \frac{r_1}{r_\alpha(P)} \frac{r_2}{r_\alpha(Q)} \min(r_\alpha(P), r_\alpha(Q))|\}$

3.2.5 Language - Operational Semantics

We present the summary of the formal definition of the operational semantics of PEPA in Figure 3.1.
Those operational rules have the following meaning:
we can infer the transition below the inference line only if the transition(s) above that line can be inferred.
The activities which a component can perform are outlined by the rules, and whenever an activity completes, it causes a transition in the system.
Moreover, there is no explicit representation of time in these rules but we assume that, for all rules, an activity will take some time to complete. In this way, some advance of time is represented by each transition. In these rules, we assume that each activity is (time) homogeneous, this mean that rates and types of activities are time-independent with respect of when they occur. Consequently we also assume that $\mathcal{A}ct(P)$, namely the activity set of a component, is time independent, and it is independent from the time at which it is considered.
The only comment presented is on the third rule for cooperation, which defines shared activities. Furthermore, the apparent rate of shared actions type in the component $E \underset{L}{\bowtie} F$, when $\alpha \in L$, is chosen such that it is the slowest of the apparent rates of that action type in both E and F. In general, we assume also that both cooperation components have to complete some work to complete the shared activity, as a reflection of their own version of that activity.
If one component has an unspecified apparent rate, other components will completely determine that rate. Moreover, to represent different possibilities of outcome, we can use multiple instances of the same action type in a component. It is also assumed that there is independence between the choice of each shared activity rate, made in order to keep the same outcome probability of each component, and the choice of outcome that each of the cooperating components makes.
Considering for example an instance (α, r_1) of action type $\alpha \in \mathcal{A}ct(E)$ and another instance (α, r_2) of action type $\alpha \in \mathcal{A}ct(F)$. When an α-type activity occurs, the probability, that it is (α, r_1), is $r_1/r_\alpha(E)$ and on the other hand the probability, that it is (α, r_2), is $r_2/r_\alpha(F)$. Instead, if α is a shared type activity in $E \underset{L}{\bowtie} F$, when it occurs, the probability that E and F combine, to form the shared activity, will be $r_1/r_\alpha(E) \times r_2/r_\alpha(F)$.

The activity rate of any activity instance, is the product of the probability of this instance completion (assuming that an activity of this type occurs) and the apparent rate in this component of the action type. Consequently there is the following rule:

Prefix

$$\frac{}{(\alpha, r).E \xrightarrow{(\alpha,r)} E}$$

Choice

$$\frac{E \xrightarrow{(\alpha,r)} E'}{E + F \xrightarrow{(\alpha,r)} E'} \qquad \frac{F \xrightarrow{(\alpha,r)} F'}{E + F \xrightarrow{(\alpha,r)} F'}$$

Cooperation

$$\frac{E \xrightarrow{(\alpha,r)} E'}{E \underset{L}{\bowtie} F \xrightarrow{(\alpha,r)} E' \underset{L}{\bowtie} F} (\alpha \notin L) \qquad \frac{F \xrightarrow{(\alpha,r)} F'}{E \underset{L}{\bowtie} F \xrightarrow{(\alpha,r)} E \underset{L}{\bowtie} F'} (\alpha \notin L)$$

$$\frac{E \xrightarrow{(\alpha,r_1)} E' \quad F \xrightarrow{(\alpha,r_2)} F'}{E \underset{L}{\bowtie} F \xrightarrow{(\alpha,R)} E' \underset{L}{\bowtie} F'} (\alpha \in L) \quad \text{where } R = \frac{r_1}{r_\alpha(E)} \frac{r_2}{r_\alpha(F)} \min(r_\alpha(E), r_\alpha(F))$$

Hiding

$$\frac{E \xrightarrow{(\alpha,r)} E'}{E/L \xrightarrow{(\alpha,r)} E'/L} (\alpha \notin L) \qquad \frac{E \xrightarrow{(\alpha,r)} E'}{E/L \xrightarrow{(\tau,r)} E'/L} (\alpha \in L)$$

Constant

$$\frac{E \xrightarrow{(\alpha,r)} E'}{A \xrightarrow{(\alpha,r)} E'} (A \stackrel{def}{=} E)$$

Figure 3.1: Operational Semantics of PEPA

$$\frac{E \xrightarrow{(\alpha,r_1)} E' \quad F \xrightarrow{(\alpha,r_2)} F'}{E \underset{L}{\bowtie} F \xrightarrow{(\alpha,R)} E' \underset{L}{\bowtie} F'} (\alpha \in L) \quad \text{where } R = r = \frac{r_1}{r_\alpha(E)} \frac{r_2}{r_\alpha(F)} \min(r_\alpha(E), r_\alpha(F))$$

Generally speaking, a set of states S, a set of transition labels T and a transition relation $\xrightarrow{t} \subseteq S \times S \quad \forall t \in T$ can define a *labelled transition system* $(S, T, \{\xrightarrow{t} | t \in T\})$. Instead, if we consider a multi-transition system, we have to replace the relation with a multi-relation whose instances reflect the number of transition between states. Taking those semantic rules, we can define PEPA as a labelled multi-transition system $(\mathcal{C}, \mathcal{A}ct, \{\xrightarrow{(\alpha,r)} | (\alpha, r) \in \mathcal{A}ct\})$, with $\mathcal{A}ct$ as set of activities, $\mathcal{C}$ as the set of components and rules in Figure 3.1 give

the multi-relation $\xrightarrow{(\alpha,r)}$.

3.2.6 Language - Additional Definitions

In this section we will present some auxiliary definitions related to PEPA, resulting from the last section. If we consider a graph in which nodes represent the language terms and arcs represent the possible transitions between them, then operational rules will define how the graph is formed. Considering also that we have already distinguished different instances of the same activity, then the graph will be a multi-graph. In this way we will distinguish different instances of an arc between terms.
This underlying derivation graph describes the possible behaviour of any component of PEPA. It provides also a useful way to reason about the model behaviour.
Finally, these are the formal notions of derivatives:

One-step derivative: *P' is a one-step derivative of P, if* $P \xrightarrow{(\alpha,r)} P'$

Derivative: *more in general, P' is a derivative of P, if* $P \xrightarrow{(\alpha_1,r_1)} \ldots \xrightarrow{(\alpha_n,r_n)} P'$
Those derivatives represent the various states of the labelled multi-transition system. It is usually convenient also to expand those definitions of a component and name each derivative individually. In this way we can define recursively the set of all possible derivatives (behaviour) of all PEPA components, in which they can evolve.
Derivative Set: *$ds(C)$ is the notation of the derivative set of a PEPA component, and it is the smallest set of components such that*:

- if $C \stackrel{def}{=} C_0$ then $C_0 \in ds(C)$;
- if $C_i \in ds(C)$ and $\exists a \in \mathcal{A}ct(C_i)$ such that $C_i \xrightarrow{a} C_j$, then $C_j \in ds(C)$.

Each reachable state of the system is captured by the derivative set of components. In this way we can visualize all possible states of the system and the relationships among them with the transition graph of a system. In fact the *derivation graph* is defined in terms of the derivative set of a system.
Derivation Graph: *the derivation graph $\mathcal{D}(C)$ of a component C and its derivative set $ds(C)$, is the labelled directed multi-graph with a set of nodes $ds(C)$ and with multiset of arcs A defined as*:

- *The elements of A given by the set $ds(C) \times ds(C) \times \mathcal{A}ct$;*
- *$\langle C_i, C_j, a\rangle$ occurs in A with the same multiplicity as the number of distinct inference trees which infer $C_i \xrightarrow{a} C_j$.*

We can denote the derivation graph and derivative set of component expressions with E, $ds(E)$ and $\mathcal{D}(E)$.
The variables in the expression are represented by leaves of the derivation graph. When a variable is instantiated, the appropriate derivation graph will be attached at that point.
$\vec{\mathcal{A}}(C)$ is the notation of the complete set of action types, used in the derivation graph of a system. This complete set represents all possible action types which can be completed during a component evolution.
Complete Set of Action Types: *the complete set of action type of a component C is*:

$$\vec{\mathcal{A}}(C) = \bigcup_{C_i \in ds(C)} \mathcal{A}(C_i)$$

3.2.7 The Underlying Stochastic Model

A stochastic process, as a representation of the system, can be built using PEPA and its derivation graphs. Furthermore, the resulting stochastic model is a continuous time Markov process, if we assume that the duration of all activities are exponentially distributed random variables.
With the CTMC, we can compute its numerical solution when it is assumed that there exists a steady-state solution. There is also a relationship between the ergodicity of the Markov process and the structure of the PEPA models.

3.2.7.1 Markov Process Generation

The generation of the underlying stochastic process can be based on the derivation graphs, for any finite model of PEPA. We know that in any model there is a component which defines it as its initial node and that its derivation graph is a multigraph. Thus, we have the following characteristics of a derivation graph:

- Initial node
- Other nodes in the graphs represent each subsequent derivative (or component)
- Between the corresponding components, for each possible transition there is an action type and an activity rate which label all arcs between nodes

To build the stochastic process we make the following associations:

- Each node of the graph is associated with a state
- Arcs of the graph define the transitions between states

Remember that in the derivation graph, the number of nodes is finite because we have assumed that also the model is finite. Moreover, in the derivation graph, the sum of all the activity rates, which label the arcs linking two nodes, represents the total transition rate between the corresponding states, since each activity duration is exponentially distributed.
We can summarize it in the following theorem:

Theorem
For any finite PEPA model $C \stackrel{def}{=} C_0$, if we define the stochastic process $X(t)$, such that $X(t) = C_i$ represents that at time t, the system behaves as component C_i, then $X(t)$ is a Markov process.

The proof of this theorem can be found in [70].

3.2.7.2 Definitions on the Markov Process Underlying a PEPA Model

There are some notions associated with the underlying Markov processes of a PEPA Model.

Sojourn Time: In a component C, the sojourn time is an exponentially distributed random variable. Moreover, C enables some activities and the sum of these activity rates represents the parameter in the distribution. Then the expected (i.e., the mean) sojourn time is:

$$\left(\sum_{a \in \mathcal{A}ct(C)} r_a \right)^{-1}$$

Exit Rates: The related notion of the exit rate from C is more convenient to consider. This rate can represent the rate at which an arbitrary activity is completed in the component C (i.e., it does something). The exit rate can also represent the rate at which the system leaves the corresponding state of component C for another one.
We denote it as $q(C)$ and its definition is:

$$q(C) = \sum_{a \in \mathcal{A}ct(C)} r_a$$

Transition Rates: The transition rate is the rate at which transitions occur between state C_i and C_j or the system changes from behaving like component C_i to behaving like component C_j. In the derivation graph there are arcs connecting the node corresponding to C_i with the node corresponding to C_j, the sum of the activity rates labeling those arcs is represented by this rate.
We denote as $q(C_i, C_j)$ the transition rate between two components C_i and C_j and so it is:

$$q(C_i, C_j) = \sum_{a \in \mathcal{A}ct(C_i|C_j)} r_a$$

Where:
$\mathcal{A}ct(C_i|C_j) = \{|a \in \mathcal{A}ct(C_i)|C_i \xrightarrow{a} C_j|\}$

Usually the previous multiset contains only a single element. Moreover, if the set of one-step derivatives of C_i doesn't contain C_j, then $q(C_i, C_j) = 0$.
The off-diagonal elements of the infinitesimal generator matrix of the Markov process Q are represented by $q(C_i, C_j)$ which can be also denoted with q_{ij}. In this way:

$$\Pr(X(t + \delta t) = C_j | X(t) = C_i) = q(C_i, C_j)\delta t + o(\delta t)$$

With: $i \neq j$

The negative sum of the non-diagonal elements of each row forms the diagonals elements ($q_{ii} = -q(C_i)$).
If the steady-state probability distribution for the system $\Pi(\cdot)$ exists, we can compute it solving the following matrix equation:

$$\Pi Q = 0$$

This will be subject to the normalization condition:

$$\sum(C_i) = 1$$

Conditional Transition Rates: The conditional transition rate is the rate at which, as a result of the completion of an activity of type α, the system changes from behaving like component C_i to behaving like component C_j. In the derivation graph there are arcs connecting the node corresponding to C_i with the node corresponding to C_j, the sum of the activity rates labelling those arcs, which have a label of the action type α, is represented by this rate.
We denote as $q(C_i, C_j, \alpha)$ the conditional transition rate between two components C_i and

C_j using the action type α.

Conditional Exit Rates: Also the conditional exit rate is considered sometimes. The exit rate represents the rate at which the system leaves the corresponding state of component C for another one after the completion of an activity of type α.
Moreover, C enables some activities and the sum of the rates of activities of type α represents this conditional exit rate. We denote it as $q(C, \alpha)$.
Furthermore, the apparent rate of α in C is the same as the conditional exit rate of C using activities of type α:

$$q(C, \alpha) = r_\alpha(C)$$

3.3 Stochastic Automata (SA)

Many high-level specification languages for stochastic discrete-event systems are based on *Markovian process algebras* and *labelled automata* [16, 69, 70, 71, 44, 21, 97] that are characterized by powerful composition operators and timed actions whose delay is governed by independent random variables with a continuous-time exponential distribution. The expressivity of such languages allows the development of well-structured specifications and efficient analyses of both qualitative and quantitative properties in a single framework. Their semantics are given in terms of stochastic automata, an extension of labelled automata with clocks that often are exponentially distributed random variables. A stochastic concurrent automaton has an underlying continuous-time Markov chain as the common denominator of a wide set of Markovian stochastic process algebra. Stochastic automata are equipped with a *composition operation* by which a complex automaton can be constructed from simpler components. A model of such automata can be found in [88, 89] which draws a distinction between *active* and *passive* action types, and in forming the composition of automata only active/passive synchronisations are permitted. An analogue semantics is proposed for Stochastic Automata Networks in [97].

Definition 3.1. *(Stochastic Automaton (SA))* A stochastic automaton P is a tuple $(\mathcal{S}_P, \mathcal{A}ct_P, \mathcal{P}ass_P, \rightsquigarrow_P, q_P)$ where

- $\mathcal{S}_P$ is a denumerable set of states called *state space* of P,
- $\mathcal{A}ct_P$ is a denumerable set of *active* types,
- $\mathcal{P}ass_P$ is a denumerable set of *passive* types,
- τ denotes the *unknown* type,
- $\mathcal{T}_P = (\mathcal{A}ct_P \cup \mathcal{P}ass_P \cup \{\tau\})$
- $\rightsquigarrow_P \subseteq (\mathcal{S}_P \times \mathcal{S}_P \times \mathcal{T}_P)$ is a transition relation where $\forall s \in \mathcal{S}_P$, $(s, s, \tau) \notin \rightsquigarrow_P$,[1]
- q_P is a function from $\rightsquigarrow_P$ to $\mathbb{R}^+$ such that $\forall s_1 \in \mathcal{S}_P$ and $\forall a \in \mathcal{P}ass_P$, $\sum_{s_2:(s_1,s_2,a)\in\rightsquigarrow_P} q_P(s_1, s_2, a) \leq 1$.

[1]Notice that τ self-loops do not affect the equilibrium distribution of the CTMC underlying the automaton. Moreover, the choice of excluding τ self-loops will simplify the definition of automata synchronisation.

In the following we denote by $\rightarrow_P$ the relation containing all the tuples of the form (s_1, s_2, a, q) where $(s_1, s_2, a) \in \rightsquigarrow_P$ and $q = q_P(s_1, s_2, a)$. We say that $q_P(s, s', a) \in \mathbb{R}^+$ is the *rate* of the transition from state s to s' with type a if $a \in \mathcal{A}ct_P \cup \{\tau\}$. Notice that this is indeed the apparent transition rate from s to s' relative to a [70]. If a is passive then $q_P(s, s', a) \in (0, 1]$ denotes the *probability* that the automaton synchronises on type a with a transition from s to s'. Hereafter, we assume that $q_P(s, s', a) = 0$ whenever there are no transitions with type a from s to s'.
If $s \in \mathcal{S}_P$, then $\forall a \in \mathcal{T}_P$ we write:

- $q_P(s, a) = \sum_{s' \in \mathcal{S}} q_P(s, s', a)$ the sum of outcoming transition from s of type a.
- $q_P(s, s') = \sum_{a \in \mathcal{T}_P} q_P(s, s', a)$ the sum of outcoming transition from s to s'.
- $q_P(s) = \sum_{a \in \mathcal{T}_P} q_P(s, a)$ the sum of outcoming transition from s.

We say that P is *closed* if $\mathcal{P}ass_P = \emptyset$. We use the notation $s_1 \overset{a}{\rightsquigarrow}_P s_2$ to denote the tuple $(s_1, s_2, a) \in \rightsquigarrow_P$; we denote by $s_1 \xrightarrow{(a,r)}_P s_2$ (resp., $s_1 \xrightarrow{(b,p)}_P s_2$) the tuple $(s_1, s_2, a, r) \in \rightarrow_P$ (resp., $(s_1, s_2, b, p) \in \rightarrow_P$) where $a \in \mathcal{A}ct_P$, $b \in \mathcal{P}ass_P$, $r \in \mathbb{R}^+$ and $p \in (0, 1]$.

Definition 3.2. *(CTMC underlying a closed SA)* The CTMC underlying a closed stochastic automaton P, denoted $X_P(t)$, is defined as the CTMC with state space $\mathcal{S}_P$ and infinitesimal generator matrix $\mathbf{Q}$ defined as: for all $s_1 \neq s_2 \in \mathcal{S}_P$,

$$q_{s_1,s_2} = \sum_{a,r:(s_1,s_2,a,r) \in \rightarrow_P} r \,.$$

For ergodic chains, we denote the equilibrium distribution of the CTMC underlying P by $\boldsymbol{\pi}_P$.

An automaton is *irreducible* if each state can be reached by any other state after an arbitrary number of transitions; moreover a closed automaton P is *ergodic* if its underlying CTMC is ergodic.
The synchronisation operator between two stochastic automata P and Q is defined in the style of master/slave synchronisation of SANs [97] based on the Kronecker's algebra and the active/passive cooperation used in Markovian process algebra such as PEPA [70].

Definition 3.3. *(SA synchronisation)* Given two stochastic automata P and Q such that $\mathcal{A}ct_P = \mathcal{P}ass_Q$ and $\mathcal{A}ct_Q = \mathcal{P}ass_P$ we define the automaton $P \otimes Q$ as follows:

- $\mathcal{S}_{P \otimes Q} = \mathcal{S}_P \times \mathcal{S}_Q$,
- $\mathcal{A}ct_{P \otimes Q} = \mathcal{A}ct_P \cup \mathcal{A}ct_Q = \mathcal{P}ass_P \cup \mathcal{P}ass_Q$,
- $\mathcal{P}ass_{P \otimes Q} = \emptyset$,
- τ is the unknown type,
- $\rightsquigarrow_{P \otimes Q}$ and $q_{P \otimes Q}$ are defined according to the rules for $\rightarrow_{P \otimes Q}$ depicted in Table 3.1: indeed, the relation $\rightarrow_{P \otimes Q}$ contains the tuples $((s_{p_1}, s_{q_1}),(s_{p_1}, s_{q_2}), a, q)$ with $((s_{p_1}, s_{q_1}), (s_{p_1}, s_{q_2}), a) \in \rightsquigarrow_{P \otimes Q}$ and $q = q_{P \otimes Q}((s_{p_1}, s_{q_1}), (s_{p_1}, s_{q_2}), a)$.

$$\frac{s_{p_1} \xrightarrow{(a,r)}_P s_{p_2} \quad s_{q_1} \xrightarrow{(a,p)}_Q s_{q_2}}{(s_{p_1}, s_{q_1}) \xrightarrow{(a,pr)}_{P \otimes Q} (s_{p_2}, s_{q_2})} \quad (a \in \mathcal{A}ct_P = \mathcal{P}ass_Q)$$

$$\frac{s_{p_1} \xrightarrow{(a,p)}_P s_{p_2} \quad s_{q_1} \xrightarrow{(a,r)}_Q s_{q_2}}{(s_{p_1}, s_{q_1}) \xrightarrow{(a,pr)}_{P \otimes Q} (s_{p_2}, s_{q_2})} \quad (a \in \mathcal{P}ass_P = \mathcal{A}ct_Q)$$

$$\frac{s_{p_1} \xrightarrow{(\tau,r)}_P s_{p_2}}{(s_{p_1}, s_{q_1}) \xrightarrow{(\tau,r)}_{P \otimes Q} (s_{p_2}, s_{q_1})} \qquad \frac{s_{q_1} \xrightarrow{(\tau,r)}_Q s_{q_2}}{(s_{p_1}, s_{q_1}) \xrightarrow{(\tau,r)}_{P \otimes Q} (s_{p_1}, s_{q_2})}$$

Table 3.1: Operational rules for SA synchronisation

3.4 Probabilistic Automata

In the discrete time we can use the probabilistic automata. A major distinction of these automata is that between fully probabilistic and non-deterministic ones. In a fully probabilistic automaton every choice is governed by a probability distribution (over set of states or states combined with actions). The probability distribution captures the uncertainty about the next state. If we abstract away from the actions in a fully probabilistic automaton, we are left with a discrete time Markov chain. Subsequently, standard techniques can be applied to analyse the resulting Markov chains. Sometimes, the incomplete knowledge about the system behaviour cannot be represented probabilistically. In these cases we should consider more than one transition possible. We speak in this case of a non-deterministic probabilistic automaton. Non-determinism is essential for modelling scheduling freedom, implementation freedom, the external environment and incomplete information. Furthermore, non-determinism is essential for the definition of an asynchronous parallel composition operator that allows interleaving. There are two main kinds of non-deterministic choices:

- external non-deterministic choices influenced by the environment, specified by having several transitions with different labels leaving from the same state

- internal non-determinism, exhibited by having several transitions with the same label leaving from a state.

We use the term non-determinism for full non-determinism including both internal and external non-deterministic choices.

Automata types can be further grouped in several subsections reflecting their common properties. Basically, every type of probabilistic automata arises from the plain definition of a transition system with or without labels. Probabilities can then be added either to every transition, or to transitions labelled with the same action, or there can be a distinction between probabilistic and ordinary (non-deterministic) states, where only the former ones include probabilistic information, or the transition function can be equipped with a structure that provides both non-determinism and probability distributions. Two classical types of probability system are:

- the reactive model

- the generative model

In a reactive system, probabilities are distributed over the outgoing transitions labelled with the same action, while in a generative system probabilities are distributed over all outgoing transitions from a state. A motivation for making this distinction is the different treatment of actions. In a reactive system actions are treated as input actions being provided by the environment. When a reactive system receives input from the environment then it acts probabilistically by choosing the next state according to a probability distribution assigned to this input. There are no probabilistic assumptions about the behaviour of the environment. On the other hand, in a generative system, as the name suggests, actions are treated as output generated by the system. When a generative system is in a state s it chooses the next transition according to the probability distribution $\alpha(s)$ assigned to s. When the transition is chosen, the system moves to another state while generating the output action which labels this transition. Note that in a generative system there is no non-determinism present, while in a reactive system there is only external non-determinism.

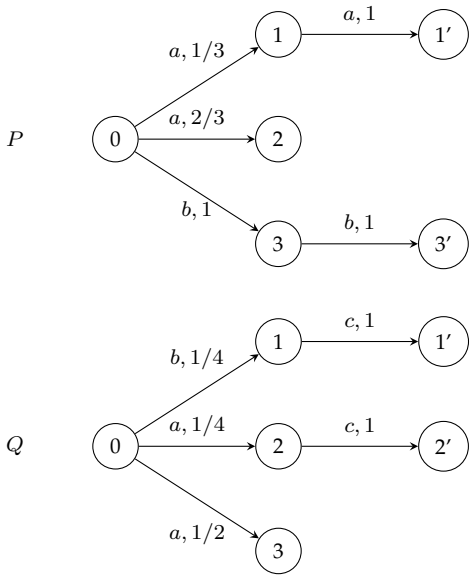


Figure 3.2: Reactive System P and generative System Q.

3.4.1 Probabilistic Input/Output Automata (PIOA)

There exists also a model which is a fusion between reactive and generative ones: *Input/Output Model.* The model of input/output probabilistic automaton, introduced by Wu, Smolka and Stark in [101], exploiting the I/O automaton by Lynch and Tuttle [82], presents a combination of the reactive and the generative model.

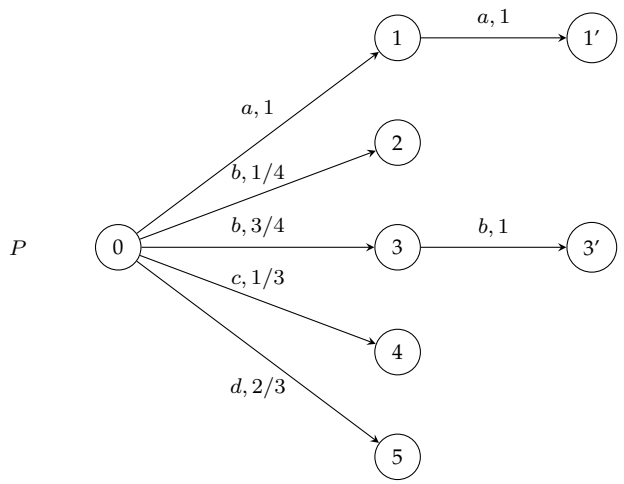


Figure 3.3: I/O System P considering: $\mathcal{A}ct_P = \{a, b\}$, $\mathcal{P}ass_P = \{c, d\}$.

In an I/O automaton for every input action there is a reactive transition. Note that the transition function for inputs is always a function and not a partial function as in the reactive models. Hence each input action is enabled in each state of an I/O probabilistic automaton. The output actions are treated generatively. At most one generative probabilistic transition gives the output behaviour of each state. We have to add also a delay rate parameter δ which is an aspect from continuous-time systems (we will discuss it later). It is obvious that, when ignoring the 0 delays, for $\mathcal{A}ct_P = \emptyset$ one gets the reactive model (with all actions enabled) and for $\mathcal{P}ass_P = \emptyset$ one gets the generative model with a delay rate assigned to each state.

The semantics of I/O automata are an extension of labelled automata with probabilities and we consider one with a delay rate parameter δ and with an underlying discrete-time Markov chain as a common denominator of a wide set of Markovian discrete process algebra. The parallel composition operation has been treated in different ways in the discrete time case:

- as unique parallel composition.
- purely synchronous (e.g., given by the product of distributions)
- asynchronous
- partly synchronous and partly asynchronous
- can strongly rely on the specific structure of the systems.

As we said, the classes of probabilistic systems can be divided into three groups dependent on whether they show reactive, generative or mixed behaviour. Classes belonging to the same of these groups allow in essence similar definition and investigation of parallel composition.
Also the probabilistic automata [101] is equipped with a *composition operation* by which

a complex automaton can be constructed from simpler components. The model draws a distinction between:

- *active* action types corresponding to *output* actions
- *passive* action types corresponding to *input* actions

Furthermore, in forming the composition of automata only active/passive synchronisations are permitted.

Definition 3.4. *(Probabilistic I/O Automaton (PIOA))* A probabilistic Input/Output automaton P is a tuple $(\mathcal{S}_P, \mathcal{A}ct_P, \mathcal{P}ass_P, \rightsquigarrow_P, q_P, \delta)$ where

- $\mathcal{S}_P$ is a denumerable set of states called *state space* of P,
- $\mathcal{A}ct_P$ is a denumerable set of *active* types,
- $\mathcal{P}ass_P$ is a denumerable set of *passive* types,
- τ denotes the *unknown* or *internal* type,
- $\mathcal{T}_P = (Act_P \cup \mathcal{P}ass_P \cup \{\tau\})$,
- $\rightsquigarrow_P \subseteq (\mathcal{S}_P \times \mathcal{S}_P \times \mathcal{T}_P)$ is a transition relation where $\forall s \in \mathcal{S}_P$ and $\forall a \in \mathcal{P}ass_P$, there exists a state $s' \in \mathcal{S}$ such that $(s, s', a) \in \rightsquigarrow_P$ [2]
- q_P is the transition probability function from $\rightsquigarrow_P$ to $(0, 1]$ which is required to satisfy the following conditions:
  - $q_P(s_1, s_2, a) > 0$ iff $(s_1, s_2, a) \in \rightsquigarrow_P$
  - $\forall s_1 \in \mathcal{S}_P$ and $\forall a \in \mathcal{P}ass_P$, $\sum_{s_2:(s_1,s_2,a)\in\rightsquigarrow_P} q_P(s_1, s_2, a)$=1.
  - $\forall s_1 \in \mathcal{S}_P$ considering $q_P(s_1, s_2, a) = 0$ if $(s, s, a) \notin \rightsquigarrow_P$, $\sum_{a\in\mathcal{A}ct_P} \sum_{s_2:(s_1,s_2,a)\in\rightsquigarrow_P} q_P(s_1, s_2, a)$=1.
- δ is the state delay function from $\mathcal{S}_P$ to $[0, \infty)$ which is required to satisfy the following conditions:
  - $\forall s \in \mathcal{S}_P$ we have $\delta(s) > 0$ iff there exists $a \in \mathcal{A}ct_P$ and $s' \in \mathcal{S}_P$ such that $(s, s', a) \in \rightsquigarrow_P$
  - $\forall s \in \mathcal{S}_P$ we have $\delta(s) = 0$ iff $\nexists a \in \mathcal{A}ct_P$ and $\nexists s' \in \mathcal{S}_P$ such that $(s, s', a) \in \rightsquigarrow_P$

The conditions on q_P can also be explained in this way:

- each transaction must have a probability greater than 0 to avoid deadlocks or unreachable states.
- the sum of all transaction of the passive type a in each state must be 1 because the input has to be always enabled and also because when they will synchronise we cannot have a probability in which we don't know what to do.
- the sum of all transactions of all active types in each state must be 1. When choosing which action to do next, we cannot have a probability greater or less than 1.

[2] This satisfies the *input-always-enabled* property for the synchronisation.

In the following we denote by $\rightarrow_P$ the relation containing all the tuples of the form (s_1, s_2, a, q) where $(s_1, s_2, a) \in \rightsquigarrow_P$ and $q = q_P(s_1, s_2, a)$. We say that $q_P(s, s', a) \in (0, 1]$ is the *probability* of the transition from state s to s' with type a. Notice that this is indeed the apparent transition rate from s to s' relative to a. Hereafter, we assume that $q_P(s, s', a) = 0$ whenever there are no transitions with type a from s to s'. We say that P is *closed* if $\mathcal{P}ass_P = \emptyset$. We use the notation $s_1 \overset{a}{\rightsquigarrow}_P s_2$ to denote the tuple $(s_1, s_2, a) \in \rightsquigarrow_P$; we denote by $s_1 \xrightarrow{(a,p)}_P s_2$ the tuple $(s_1, s_2, a, p) \in \rightarrow_P$.

Definition 3.5. *(Closed PIOA)* A probabilistic Input/Output automaton P is a tuple $(\mathcal{S}_P, \mathcal{A}ct_P, \mathcal{P}ass_P, \rightsquigarrow_P, q_P, \delta)$ and it is considered closed if $\mathcal{P}ass_P = \emptyset$.

Definition 3.6. *(DTMC underlying a closed PIOA)* The DTMC underlying a closed probabilistic automaton P, denoted $X_P(t)$, is defined as the DTMC with state space $\mathcal{S}_P$ and transition probability matrix $\mathbf{P}$ defined as: $\forall (s_1, s_2) \in \mathcal{S}_P$,

$$p_{s_1,s_2} = \sum_{a,p:(s_1,s_2,a,p)\in\rightarrow_P} p\,.$$

For ergodic chains, we denote the equilibrium distribution of the DTMC underlying P by $\boldsymbol{\pi}_P$.

An automaton is *irreducible* if each state can be reached by any other state after an arbitrary number of transitions, moreover a closed automaton P is *ergodic* if its underlying DTMC is ergodic.
Also the synchronisation operator between two probabilistic automata P and Q is defined in the style of master/slave synchronisation. In the PIOA the actions are divided into passive and active, and while there can be synchronisation on passive actions, as in the reactive setting, the sets of active actions in each of the components must be disjoint.

Definition 3.7. *(Compatible PIOAs)* Given two probabilistic Input/Output automata P and Q, they are compatible if and only if

$$\mathcal{A}ct_P \cap \mathcal{A}ct_Q = \emptyset$$

Moreover we use the following convention:
If $s \in \mathcal{S}_P$ (resp., $\mathcal{S}_Q$) and $a \in (\mathcal{P}ass_P \cup \mathcal{A}ct_Q \cup \mathcal{P}ass_Q)$ (resp., $(\mathcal{A}ct_P \cup \mathcal{P}ass_P \cup \mathcal{P}ass_Q)$) and there is no transition from s involving a, then we consider that $s \xrightarrow{(a,1)}_P s'$ (resp., $s \xrightarrow{(a,1)}_Q s'$) and if $a \notin \mathcal{P}ass_P$ then $a \in \mathcal{P}ass_P$.
In other words, if there exists an action type in the sets of Q but there is no transition from a state of P of that type, then we will consider an implied passive action $s \xrightarrow{(a,1)}_P s'$. This convention will enforce the "input-always-enabled" requirement for the composite automaton.

Synchronisation operator is only defined on *compatible* automata.

Definition 3.8. *(PIOA Synchronisation)* Given two compatible probabilistic Input/Output automata P and Q we define the automaton $P \otimes Q$ as follows:

- $\mathcal{S}_{P\otimes Q} = \mathcal{S}_P \times \mathcal{S}_Q$,
- $\mathcal{A}ct_{P\otimes Q} = \mathcal{A}ct_P \cup \mathcal{A}ct_Q$,

$$\frac{s_{p_1} \xrightarrow{(a,p)}_P s_{p_2} \quad s_{q_1} \xrightarrow{(a,q)}_Q s_{q_2}}{(s_{p_1}, s_{q_1}) \xrightarrow{(a,pq)}_{P\otimes Q} (s_{p_2}, s_{q_2})} \ (a \in \mathcal{P}ass_{P\otimes Q})$$

$$\frac{s_{p_1} \xrightarrow{(a,p)}_P s_{p_2} \quad s_{q_1} \xrightarrow{(a,q)}_Q s_{q_2}}{(s_{p_1}, s_{q_1}) \xrightarrow{(a,D_1pq)}_{P\otimes Q} (s_{p_2}, s_{q_2})} \ (a \in \mathcal{A}ct_P)$$

$$\frac{s_{p_1} \xrightarrow{(a,p)}_P s_{p_2} \quad s_{q_1} \xrightarrow{(a,q)}_Q s_{q_2}}{(s_{p_1}, s_{q_1}) \xrightarrow{(a,D_2pq)}_{P\otimes Q} (s_{p_2}, s_{q_2})} \ (a \in \mathcal{A}ct_Q)$$

$$\frac{s_{p_1} \xrightarrow{(\tau,p)}_P s_{p_2}}{(s_{p_1}, s_{q_1}) \xrightarrow{(\tau,D_1p)}_{P\otimes Q} (s_{p_2}, s_{q_1})} \qquad \frac{s_{q_1} \xrightarrow{(\tau,p)}_Q s_{q_2}}{(s_{p_1}, s_{q_1}) \xrightarrow{(\tau,D_2p)}_{P\otimes Q} (s_{p_1}, s_{q_2})}$$

$$D_1 = \frac{\delta_P(s_{p_1})}{\delta_P(s_{p_1}) + \delta_Q(s_{q_1})} \qquad D_2 = \frac{\delta_Q(s_{q_1})}{\delta_P(s_{p_1}) + \delta_Q(s_{q_1})}$$

Table 3.2: Operational rules for PIOA synchronisation

- $\mathcal{P}ass_{P\otimes Q} = (\mathcal{P}ass_P \cup \mathcal{P}ass_Q) \setminus \mathcal{A}ct_{P\otimes Q}$,
- τ is the unknown type,
- $\delta_{P\otimes Q} = \delta_P + \delta_Q$,
- $\rightsquigarrow_{P\otimes Q}$ and $q_{P\otimes Q}$ are defined according to the rules for $\rightarrow_{P\otimes Q}$ depicted in Table 3.2: indeed, the relation $\rightarrow_{P\otimes Q}$ contains the tuples $((s_{p_1}, s_{q_1}),(s_{p_1}, s_{q_2}), a, p)$ with $((s_{p_1}, s_{q_1}), (s_{p_1}, s_{q_2}), a) \in \rightsquigarrow_{P\otimes Q}$ and $p = q_{P\otimes Q}((s_{p_1}, s_{q_1}), (s_{p_1}, s_{q_2}), a)$.

We have to notice that $\mathcal{P}ass_{P\otimes Q}$ is the set of *passive* actions which remain passive because they don't synchronise with anybody. Moreover, since $\mathcal{A}ct_{P\otimes Q} = \mathcal{A}ct_P \cup \mathcal{A}ct_Q$ we have $\delta_{P\otimes Q} = \delta_P + \delta_Q$ because the *delay rate* increase (hence the waiting time for the next active action decreases) with more possible active actions which can take place.

The proof that $P \otimes Q$ is well defined in the class I/O can be found in [101]. Let us now informally explain the definition of parallel composition, and the role of the functions δ_P and δ_Q and more in general of δ. If s is a state of P, then $\delta_P(s)$ is a positive real number corresponding to the *delay rate* in state s. It is a rate of an exponential distribution, determining the time that the automaton waits in state s until it generates one of its active actions. If no active actions are enabled in this state then $\delta_P(s) = 0$. When determining the distribution on active actions for $P \otimes Q$, the components distributions are joined in one such that any probability of P is multiplied with normalization factor $\frac{\delta_P}{\delta_P + \delta_Q}$ and any probability of Q is multiplied with $\frac{\delta_Q}{\delta_P + \delta_Q}$. Note that by the compatibility assumption, no action appears both in the set of active actions of P and Q. The normalization factor models a racing policy between the states s_{P1} and s_{q1} for generating their own

active actions. The value $\frac{\delta_P(s_{p_1})}{\delta_P(s_{p_1}) + \delta_Q(s_{q_1})}$ is the probability that the state s_{p1} has less waiting time left then the state s_{q1} and therefore wins the race and generates one of its own active actions. On the other hand, synchronisation occurs on all passive actions, no autonomous behaviour is allowed by the components on active actions, corresponding to the assumption that the input of those passive states is provided by the environment or other processes and must be enabled in any state.

3.4.1.1 Example

Let's now see an example of two simple cooperating processes S and T (Table 3.4) with:

- $\delta_s(0) = 2$, $\delta_t(0) = 3$ and $\delta_{s\otimes t}(0,0) = 5$.
- $\mathcal{A}ct_S = \{d, e\}$, $\mathcal{A}ct_T = \{b\}$ and $\mathcal{A}ct_{S\otimes T} = \{b, d, e\}$
- $\mathcal{P}ass_S = \{a, b\}$, $\mathcal{P}ass_T = \emptyset$ and $\mathcal{P}ass_{S\otimes T} = (\{a, b\} \cup \emptyset) \setminus \{b, d, e\} = \{a\}$

Clearly the states s_0 and t_0 are compatible since $\mathcal{A}ct_S \cap \mathcal{A}ct_T = \emptyset$. Moreover, due to the convention *input-always-enabled* we consider $t_0 \xrightarrow{(a,1)}_T t_0$, $t_0 \xrightarrow{(d,1)}_T t_0$, $t_0 \xrightarrow{(e,1)}_T t_0$ and then we have $\mathcal{P}ass_T = \{a, d, e\}$. Their cooperation can be seen in Table 3.5.

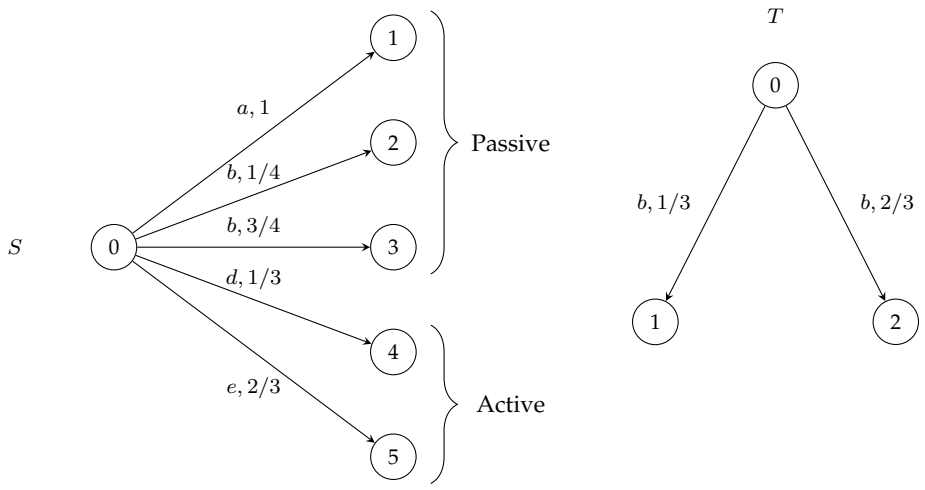
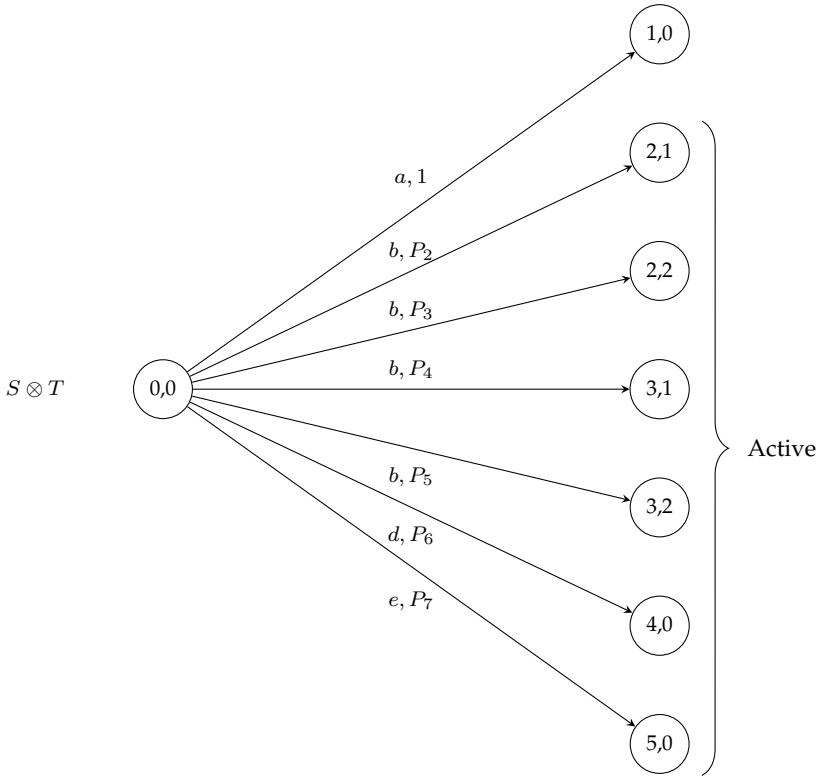


Figure 3.4: Processes S and T.

3.5 Product-forms at continuous time

The product-form property states that *the steady-state probabilities of the joint process can be expressed as the normalized product of the steady-state probabilities of its interacting components*. We can use product-form stochastic models for the performance evaluation of systems in software and hardware architectures, and communication protocols. Using several high-level formalisms, we can define product-form stochastic models. For the class of Markovian queueing networks, we can use the BCMP theorem to provide a product-form



$$P_2 = \frac{\delta_S(s_0)}{\delta_S(s_0) + \delta_T(t_0)} * \frac{1}{4} * \frac{1}{3} = \frac{3}{5} * \frac{1}{12} = \frac{1}{20}$$

$$P_3 = \frac{3}{5} * \frac{1}{4} * \frac{2}{3} = \frac{2}{20}$$

$$P_4 = \frac{3}{5} * \frac{3}{4} * \frac{1}{3} = \frac{3}{20}$$

$$P_5 = \frac{3}{5} * \frac{3}{4} * \frac{2}{3} = \frac{6}{20}$$

$$P_6 = \frac{2}{5} * \frac{1}{3} * 1 = \frac{2}{15}$$

$$P_7 = \frac{2}{5} * \frac{2}{3} * 1 = \frac{4}{15}$$

$$P_2 + .. + P_6 = \frac{1}{20} + \frac{2}{20} + \frac{3}{20} + \frac{6}{20} + \frac{2}{15} + \frac{4}{15} = \frac{12}{20} + \frac{6}{15} = 1$$

Figure 3.5: Processes S and T cooperating.

solution [14]. Moreover, we can use $M \implies M$ property and the Reversed Compound Agent Theorem (RCAT) to analyse product-form models of queueing networks (QN) and Markovian Process Algebra (MPA) [94, 61].

3.5.1 Motivations

Using product-form models, we can evaluate precise and detailed results like queue length distribution, average response time, resource utilization and throughput. These performance measures can be computed for the overall network and also for each of its components. We have to make a set of assumption on the parameters of the system in order to analyse the product-form network. This leads to a closed-form expression of the stationary state distribution.
Let's consider a queuing network with a single chain and N servers, we define its states with $\mathbf{n} = (n_1, n_2, ..., n_N)$. n_i represents the number of customers at station i; the sum of all of its states $\sum_{i=1}^{N} n_i$ is the system overall population. The associated Markov process can define a closed-form of the joint queue length distribution π of the Product-form queuing network as:

$$\pi(\mathbf{n}) = \frac{1}{G} d(n) \prod_{i=1}^{N} g_i(n_i)$$

where G is a normalizing constant, d is a function defined in terms of network parameters and function g of n_i depends on the type of server i. $G = 1$ for open networks, on the contrary $d(n) = 1$ for closed networks. Jackson introduced product-forms for open queuing networks [74] and for closed queuing networks by Gordon and Newell [58]. For these models, we have to consider only single-class and single-chain queuing networks. Moreover, we have to require exponential service time distributions and for Jackson networks also Poisson arrivals. The BCMP Theorem [14] considers non-exponential service time distributions for certain scheduling disciplines and extends these classes of queuing networks to open, closed, mixed, multi-class and multiple chain queuing networks.
We can efficiently analyse product-form queuing networks using algorithms with a polynomial time computational complexity in the number of their components. In this way, we can achieve a good balance between a relatively high accuracy in the performance results and the efficiency in model analysis and evaluation for this class of models. The application of this class as a powerful tool for performance evaluation is achieved thanks to the satisfaction of several interesting properties such as insensitivity and exact aggregation by product-form networks. The class of BCMP queuing networks and some properties of product-form will be now briefly defined.

3.5.2 BCMP Theorem

BCMP theorem [14] is named after its authors Baskett, Chandy, Muntz and Palacios. It characterizes a wide class of queuing networks with product-form distribution.

Model Definition

The queuing network consists in N service stations $\Omega = \{1, 2, .., N\}$ and the number of classes R and chains C are the same $R = C$. There can be 4 service disciplines:

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

2. Processor Sharing (PS) discipline,

3. Infinite Server (IS) discipline,

4. Last Come First Serviced (LCFS) with Preemptive Resume (also known as LCFSPR)

Except for FCFS we have to assume that the average service rate can depend on the state of each customer chain and the service time distributions have rational Laplace transforms.
Moreover we have that:

- $\mu_i^{(j)}$ is the service rate of server i for chain j customers. For FCFS $\mu_i^{(j)} = \mu_i$ and the service time is independent from the number of customers.
- $\mathbf{n} = (\mathbf{n_1}, \mathbf{n_2}, ..., \mathbf{N})$ is the state of system
- $\mathbf{n_i} = (n_i^{(1)}, n_i^{(2)}, ..., n_i^{(C)})$ is the occupancy vector of server i
- $n_i = \sum_{c=1}^{C} n_i^{(c)}$ is the total number of customers in i server
- $n = \sum_{k=1}^{N} n_k$ is the total number of customers in the system
- $n^{(j)} = \sum_{k=1}^{N} n_k^{(j)}$ is the total number of customers of chain j in the system.
- when we have an open system, customers can arrive in the network from an external source (e.g., another unidentified system) with a Poisson process with parameter λ_n which depends on the number of customers n in the system or with parameter $\lambda_i(j)$ which depends on the number of customers j in queue i.

BCMP theorem, single class, multiple chain [14]

Considering Ω a BCMP queuing network under stability conditions. Then we have that the steady-state probability holds and its equal to:

$$\pi(n) = \frac{1}{G} d(\mathbf{n}) \prod_{i=1}^{N} g_i(\mathbf{n_i})$$

where $d(\mathbf{n}) = \prod_{a=0}^{n-1} \lambda(a)$ if arrival rate depends on the total number of customers in the network or $d(\mathbf{n}) = \prod_{j=0}^{C} \prod_{a=0}^{n^{(j)}-1} \lambda_j(a)$, otherwise. $g_i(\mathbf{n_i})$ functions are determined with respect of service discipline:

- $g_i(\mathbf{n_i}) = n_i! \left[\prod_{c=1}^{C} \frac{1}{n_i^c!} (\rho_i^{(c)})^{n_i^{(c)}} \right]$, for FCFS, PS and LCFSPR disciplines.
- $g_i(\mathbf{n_i}) = \prod_{c=1}^{C} \frac{1}{n_i^c!} (\rho_i^{(c)})^{n_i^{(c)}}$, for IS discipline.

For what concerns the BCMP theorem for multi-class and multiple chain queuing networks, the theorem of steady-state probability still holds but using different definitions for g_i.

3.5.2.1 Extensions of the BCMP class

We can find some extension of systems in BCMP product-form to define formalisms to model more complex systems with more features (e.g., dynamic load balancing algorithms, adaptive routing strategies). These extensions include systems with:

- State-dependent routing in which routing probabilities of customers can depend on the state of the entire system or on the single queue in [19, 79, 106]
- Different service disciplines, e.g., multiple-server nodes with concurrent classes of customers in [20].
- Finite capacity queues in which the system has sub-network constraints and blocking policies deciding the behaviour of customer arrivals and server activities when the queue is at its full capacity. These kinds of queues have a product-form solution in some special case [2, 59, 9, 109].
- Batch of arrivals and batch of services, related also to discrete-time queuing networks. These systems' behaviours are described by discrete-time Markov chains assuming special expression in the case of batch of arrivals and departures. The quasi-reversibility property and the generalized expressions for traffic equations are the bases of the product-form solution which holds for both discrete and continuous time [67, 68].
- Also G-Networks [48] (described in section 2.6) with product-form solutions are an extension of queuing networks. These product-form solutions are based on sets of non-linear traffic equations of the customers and have exponential and independence assumptions. These queues can be used to model some special system behaviours [51], dealing with also multi-class of customers [37], with customers' resets [36] and with triggered batch signal movement, state-dependent service rate and routing intensities [38].

3.5.3 Characterization of Queuing Networks in Product-Forms

We can derive a set of performance indices without generating and solving the underlying Markov processes and the system of global balance equations if the system is in product-form. Under some assumptions (e.g., infinite queue capacity, non-blocking factors, non-priority scheduling, state-independent routing probabilities), we can determine whether a well-formed queuing network has a BCMP product-form solution, giving conditions on server queuing discipline and on service time distributions. *Local balance property, $M \implies M$ property and quasi-reversibility property* are all strictly related to product-form.

3.5.4 Local Balance Property

The local balance property states that *the effective rate at which the system leaves state s after a service completion of a chain j customer at station i, equals the effective rate at which the system enters state s due to an arrival of chain j customer to station i.* In [95, 26] we see that local balance holds even when service time distributions are represented by a network of exponential stages but we must track the stage at which a customer is being served.
Moreover, we can notice that:

- local balance equations (LBEs) are a sufficient but not necessary condition for the system solution. In fact, if the steady-state probability distribution π satisfy the LBEs then also the global balance equations (GBEs) are satisfied, but the opposite is not true.
- it's computationally more efficient to solve LBEs than GBEs even if we have still to handle the set of reachable states. It can be a problem for open chains or networks. Moreover, it's more simple to prove that the steady-state formula is correct checking if it verifies LBEs than GBEs.
- The local balance is a property of a station embedded in a queuing network. Nevertheless, we are considering states that are still part of the network.

3.5.5 $M \implies M$ property

Introduced in [94], $M \implies M$ property is defined for a single queueing system. Open queuing networks must be under independent Poisson process condition for both arrivals and departures for each class of customers in order to satisfy this property.
Let us define a queue in isolation with state space S and R classes. λ_r is the independent Poisson processes rate of customer arrivals of class r. $\pi(s)$ is the steady-state probability of state s with $s \in S$ and $|s|_r$ is the number of customers of class r in the queue when the system is in state s. The $M \implies M$ property holds if we have that:

$$\forall s \in S \qquad \prod_{s' \in S_r^+} \frac{\pi(s') q_{s's}}{\pi(s)} = \lambda_r$$

with $S_r^+ = \{s' : |s'|_r = |s|_r + 1\}$ and $q_{s's}$ represents the transition rate between state s' and s.
We can notice that:

- The queue is considered in isolation in the $M \implies M$ property. In this way, we can decide if we can embed the queue, with specific service time distribution and queuing discipline, into a product-form queuing network ([20, 1]). If the $M \implies M$ property is satisfied by all stations of a queuing network then the system has a product-form solution.
- If we have an open system and all stations satisfy the $M \implies M$ property, then the entire system fulfils the $M \implies M$ property [94].
- In a queuing network with a server using a non-priority scheduling discipline, all stations satisfy $M \implies M$ property if and only if the local balance property holds.

3.6 Quasi reversibility

In a system, *if the queue at time t has a length independent of the departure times of customer prior to time t and of the arrival times of customers after t*, then the queuing system has the *quasi-reversibility property*. In a queuing network, if all its stations are quasi-reversible then it has also a product-form solution. We can find a proof of this in [76].
Moreover, we have to notice that:

- The quasi-reversibility property is only defined for the queue in isolation.

- In a quasi-reversible system, both arrival and departure streams should be independent and Poisson (A proof can be found in [75]). In this way, a system is quasi-reversible if and only if it exhibits the $M \implies M$ property.

3.6.1 Quasi-Reversible Automata

Now we review the definition of quasi-reversibility given by Kelly in [76] by using the notation of stochastic automata of Section 3.3. In order to clarify the exposition, we introduce a closure operation over stochastic automata that allows us to assign to all the transitions with the same passive type the same rate λ.

Definition 3.9. *(SA closure)* The closure of a stochastic automaton P with respect to a passive type $a \in \mathcal{P}ass_P$ and a rate $\lambda \in \mathbb{R}^+$, written $P^c = P\{a \leftarrow \lambda\}$, is the automaton defined as follows:

- $\mathcal{S}_{P^c} = \{s^c \mid s \in \mathcal{S}_P\}$
- $\mathcal{A}ct_{P^c} = \mathcal{A}ct_P$ and $\mathcal{P}ass_{P^c} = \mathcal{P}ass_P \setminus \{a\}$
- $\rightsquigarrow_{P^c} = \{(s_1^c, s_2^c, b) |\ (s_1, s_2, b) \in \rightsquigarrow_P,\ a \neq b\} \cup \{(s_1^c, s_2^c, \tau) |\ (s_1, s_2, a) \in \rightsquigarrow_P\}$
- $$q_{P^c}(s_1^c, s_2^c, b) = \begin{cases} q_P(s_1, s_2, b) & \text{if } b \neq a, \tau \\ q_P(s_1, s_2, a)\lambda + q_P(s_1, s_2, \tau) & \text{if } b = \tau \end{cases}$$
  where we assume that $q_P(s_1, s_2, b) = 0$ if $(s_1, s_2, b) \notin \rightsquigarrow_P$.

Notice that for a closure P^c of a stochastic automaton P with respect to all its passive types in $\mathcal{P}ass_P$ we can compute the equilibrium distribution, provided that the underlying CTMC is ergodic (see Definition 3.2).

Definition 3.10. *(Quasi-reversible SA [76, 90])* An irreducible stochastic automaton P with $\mathcal{P}ass_P = \{a_1, \ldots, a_n\}$ and $\mathcal{A}ct_P = \{b_1, \ldots b_m\}$ is quasi-reversible if

- for all $a \in \mathcal{P}ass_P$ and for all $s \in \mathcal{S}_P$, $\sum_{s' \in \mathcal{S}_P} q_P(s, s', a) = 1$
- for each closure $P^c = P\{a_1 \leftarrow \lambda_1\} \ldots \{a_n \leftarrow \lambda_n\}$ with $\lambda_1, \ldots, \lambda_n \in \mathbb{R}^+$ there exists a set of positive real numbers $\{k_{b_1}, \ldots, k_{b_m}\}$ such that for each $s \in \mathcal{S}_{P^c}$ and $1 \leq i \leq m$
$$k_{b_i} = \frac{\sum_{s' \in \mathcal{S}_{P^c}} \pi_{P^c}(s') q_{P^c}(s', s, b_i)}{\pi_{P^c}(s)}, \tag{3.1}$$
  where $\boldsymbol{\pi}_{P^c}$ denotes any non-trivial invariant measure of the CTMC underlying P^c.

Notice that in the definition of quasi-reversibility we do not require the closure of P with respect to all its passive types to give rise to a stochastic automaton with an ergodic underlying CTMC because we assume $\boldsymbol{\pi}_{P^c}$ to be an invariant measure, i.e., we do not require that $\sum_{s \in \mathcal{S}_{P^c}} \pi_{P^c}(s) = 1$. However, the irreducibility of the CTMC underlying the automaton ensures that all the invariant measures differ for a multiplicative constant, hence Equation (3.1) is independent of the choice of the invariant measure.
The next theorem states that a network of quasi-reversible stochastic automata exhibits a product-form invariant measure and, if the joint state space is ergodic, a product-form equilibrium distribution. For the sake of simplicity, we state the theorem for two synchronising stochastic automata although the result holds for any finite set of automata which synchronise pairwise [76, 61, 90].

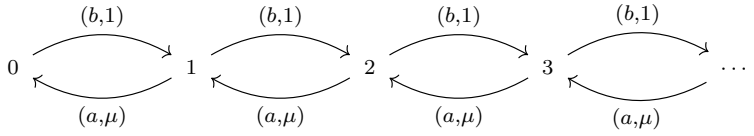


Figure 3.6: Stochastic automaton underlying a Jackson's queue.

Theorem 3.11. (Product-form solution based on quasi-reversibility) *Let P and Q be two quasi-reversible automata such that $\mathcal{A}ct_P = \mathcal{P}ass_Q$ and $\mathcal{A}ct_Q = \mathcal{P}ass_P$ and let $S = P \otimes Q$. Assume that there exists a set of positive real numbers $\{k_a : a \in \mathcal{A}ct_P \cup \mathcal{A}ct_Q\}$ such that if we define the following automata $P^c = P\{a \leftarrow k_a\}$ for each $a \in \mathcal{P}_P$ and $Q^c = Q\{a \leftarrow k_a\}$ for each $a \in \mathcal{P}_Q$ it holds that:*

$$k_a = \frac{\sum_{s' \in \mathcal{S}_{P^c}} \pi_{P^c}(s') q_{P^c}(s', s, a)}{\pi_{P^c}(s)} \qquad \forall s \in \mathcal{S}_{P^c}, a \in \mathcal{A}ct_P$$

$$k_a = \frac{\sum_{s' \in \mathcal{S}_{Q^c}} \pi_{Q^c}(s') q_{Q^c}(s', s, a)}{\pi_{Q^c}(s)} \qquad \forall s \in \mathcal{S}_{Q^c}, a \in \mathcal{A}ct_Q$$

Then, given the invariant measures $\boldsymbol{\pi}_{P^c}$ and $\boldsymbol{\pi}_{Q^c}$ it holds that

$$\pi_S(s_1, s_2) = \pi_{P^c}(s_1^c)\pi_{Q^c}(s_2^c)$$

is an invariant measure for all the positive-recurrent states $(s_1, s_2) \in \mathcal{S}_S$ where s_1^c and s_2^c are the states in $\mathcal{S}_{P^c}$ and $\mathcal{S}_{Q^c}$ corresponding to $s_1 \in \mathcal{S}_P$ and $s_2 \in \mathcal{S}_Q$, according to Definition 3.9. In this case we say that P and Q have a quasi-reversibility based product-form.

Example 1. *(Product-form solution of Jackson networks)* Jackson networks provide an example of models having a product-form solution. A network consists of a collection of exponential queues with state-independent probabilistic routing. Jobs arrive from the outside at each queuing station in the network according to a homogeneous Poisson process. It is well-known that the queues of Jackson networks are quasi-reversible and hence the product-form is a consequence of Theorem 3.11. Figure 3.6 shows the automaton underlying a Jackson's queue where a is an active type while b is a passive one. It is worth noticing that also the queues considered in [14, 20] are quasi-reversible. □

3.7 Product-Forms in PEPA

Using the PEPA language, we can reformulate the product-form property. Let's define P and Q as two interacting components. If we can express the stationary probability distribution of their cooperation as a product function depending only on the states of P and Q in isolation, then the system is in product-form. We can classify these product-form solutions with respect to:

- Models with a reversible CTMC [72];
- Models with a quasi-reversible CTMC [64];
- Models based on RCAT theorem and its extensions [61, 63]

Moreover, besides these general classifications, there are also other kinds of reformulations for the product-form property using the PEPA language. Models belonging to the class of competing Markov processes identified by Boucherie [18], are analysed using PEPA models in [73]. Another product-form property for PEPA, derived from stochastic Petri nets, are presented in [100]. Furthermore, in [28], we can see an extension of PEPA which guarantee the product-form using a derived combinator for constructing process algebra models.

3.7.1 Reversible Models

This product-form solution is based on the reversibility property of CTMC. In this solution we have complementary types of activities α and $-\alpha$. α and $-\alpha$ are a reverse pair because if we can leave a state due to an α activity then in the arrival state there must be an $-\alpha$ activity which allows us to return back to the previous state. These syntactical conditions are quite strict but they ensure that a PEPA model has a product-form solution [72].

3.7.2 Quasi-Reversible Models

Quasi-reversible models are based on the same main idea of reversible ones. A component P enables a reverse pair $(\alpha, -\alpha)$ if there is an enabled activity (a, r) in P and for each of its derivative components P' such that $P \xrightarrow{(\alpha,r)} P'$ there exists an activity $P' \xrightarrow{(-\alpha,s)} P$, with r and s positive real numbers or $\top$. A reverse pair can be associated with customers' arrivals and departures. A PEPA component is an input/output component if it enables only two activities forming a reverse pair: passive $(\alpha, \top)$ and active $(-\alpha, r)$, with $r \in \mathbb{R}^+$. The analysis of these models [64] has two main steps:

1. A PEPA component with a quasi-reversible underlying CTMC is called a QR-component. A subset of QR-components, called input/output components, can be syntactically recognized and these have the same role as queueing stations in a product-form queueing systems. In this way we can check if a PEPA component is an input/output component without generating its underlying CTMC and its derivation graph.

2. More complex components of the system are studied as combinations of QR-components. In [64] a set of sufficient syntactical conditions are defined to ensure that the CTMC of the whole system is still quasi-reversible. There is a distinction between open and closed interaction which can be associated to open and closed queuing networks, respectively. This result holds also for the class of QR-component models and not just for input/output systems.

3.7.3 Reversed Compound Agent Theorem

One of the most interesting results in the Markovian process algebras is the Reversed Compound Agent Theorem (RCAT) [61] and its extension [60, 63]. This theorem includes other product-form models such as:

- Boucherie product-forms
- G-Networks

- Jackson queuing networks product-forms
- Coleman, Henderson et al. product-forms

The main idea of the theorem is that RCAT derives the steady-state probabilities of two interacting components P and Q, analysing the reversed processes of the two components (S and R, respectively). Replacing in P and Q their occurrence of the passive action type transitions with rates that can be algorithmically calculated we obtain R and S.

Reversed actions of multiple actions

The reversed actions of multiple actions a_i, λ_i with $1 \leq i \leq n$ that an agent P can perform, which lead to the same derivative Q are respectively:

$$(\overline{a_i}, (\lambda_i/\lambda)\overline{\lambda})$$

where $\lambda = \sum_{i=1}^{n} \lambda_i$ is the sum of forward rates and $\overline{\lambda}$ is the reversed rate of the (composite) transition in the CTMC with rate λ corresponding to all the arcs between P and Q.

RCAT and its extensions only deal with cooperation where actions are active with a specified rate or passive with $\top$ rate. Let's define a cooperation $P \underset{L}{\bowtie} Q$, we denote as:

- $\mathcal{P}_P(L)$ the passive set of P with action types of L;
- $\mathcal{A}_P(L)$ the active set of P with action types of L;

We have that $\mathcal{P}_P(L) \cup \mathcal{A}_P(L) = \mathcal{P}_Q(L) \cup \mathcal{A}_Q(L) = L$. An action type a is enabled in a component if it can carry out an activity with type a.

RCAT Theorem

Assuming that $P \underset{L}{\bowtie} Q$ has an irreducible derivation graph. If the following conditions hold:

1. every passive action in $\mathcal{P}_P(L)$ or $\mathcal{P}_Q(L)$ is always enabled in P or Q (i.e. enabled in all the states of the transition graph);
2. every reversed action of an active action type in $\mathcal{A}_P(L)$ or $\mathcal{A}_Q(L)$ is always enabled in $\overline{P}$ or $\overline{Q}$;
3. every occurrence of a reversed action of an active action type in $\mathcal{A}_P(L)$ (or $\mathcal{A}_Q(L)$) has the same rate in $\overline{P}$ (or $\overline{Q}$).

Then we have that the reversed agent $\overline{P \underset{L}{\bowtie} Q}$ has the following derivation graph:

$$\overline{R}\{(\overline{\alpha}, \overline{p_\alpha}) \leftarrow (\overline{\alpha}, \top) | \alpha \in \mathcal{A}_P(L)\} \underset{L}{\bowtie} \overline{S}\{(\overline{\alpha}, \overline{q_\alpha}) \leftarrow (\overline{\alpha}, \top) | \alpha \in \mathcal{A}_Q(L)\}$$

where:

- $\leftarrow$ represent a syntactical substitution of the left hand side part with the right hand side part in the component definition;
- $R = P\{\top_a \leftarrow x_\alpha | \alpha \in \mathcal{P}_P(L)\}$ and $S = Q\{\top_a \leftarrow x_\alpha | \alpha \in \mathcal{P}_Q(L)\}$;
- $\{x_\alpha\}$ are the solutions of the equations for $\top_\alpha$:

$$\top_\alpha = \overline{q_\alpha} \qquad \alpha \in \mathcal{P}_P(L)$$
$$\top_\alpha = \overline{p_\alpha} \qquad \alpha \in \mathcal{P}_Q(L)$$

- $\overline{p_\alpha}$ and $\overline{q_\alpha}$ are the symbolic rates of action type $\overline{\alpha}$ in P and Q.

3.8 Limitations of RCAT and Quasi-reversible Product-forms

The class of product-forms is wide and includes different kinds of system. Even if the Reversed Compound Agent Theorem (RCAT) [61] and Extended Reversed Compound Agent Theorem (ERCAT) [63] are very general results for the analysis of product-form stochastic models, they have some limits and do not capture all product-forms models. In fact, some systems have a product-form solution, even if they do not satisfy the conditions of these theorems. In [8] we can see a method for analysing Markov modulated processes as well as other product-form model classes, opportunely formulated in order to apply ERCAT theorem. These include quasi-reversible queueing networks [76], G-Networks with various types of triggers [62], queueing networks with finite capacity and blocking [6], stochastic Petri nets [10, 84] and others. However there are some examples of meaningful models whose product-form cannot be analysed by applying these results. As far as we know, the methodology used in RCAT and in its extensions is sufficient but not necessary for product-forms. For now, there is not a general rule for product-form solutions thus we cannot state that RCAT it's not just an incomplete version of the general rule (if it will ever exists) even if its main idea could be used to search for new product-forms. We will show now an example of a product-form Markov modulated process whose stationary distribution cannot be derived via ERCAT.
In general a system of two tandem queues in which the service rate of the second depends on the number of customers of the first does not have a product-form stationary distribution. Moreover, as analysed in [14, 76], it is not considered in the state-dependent service rate functions. Under some assumptions, the product-form exists even if it is not derivable by previous known results.

3.8.1 Example of Product-form not satisfying ERCAT

Let's consider two tandem queues P and Q as depicted in Figure 3.7. Customers arrive at P according to a homogeneous Poisson process with rate λ and at Q according to a Poisson process whose rate $\lambda_D(e)$ depends on the number of customers e present in queue P. We assume that:

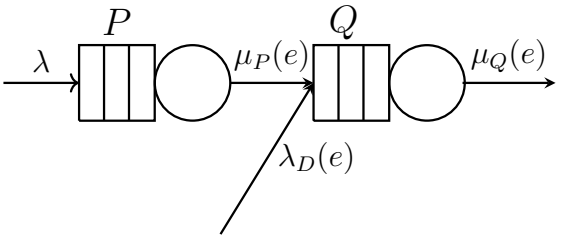


Figure 3.7: Tandem of two queues.

$$\lambda_D(e) = \begin{cases} \lambda_H & \text{if } 0 \leq e < n \\ \lambda_L & \text{if } e \geq n \end{cases}$$

for $n > 0$. Moreover, P has an exponentially distributed service time whose rate $\mu_P(e)$ depends on the number of customers e in P. Q has an exponentially distributed service

time whose rate depends on the number of customers e in P according to the following definition:

$$\mu_Q(e) = \begin{cases} \mu_H & \text{if } 0 \leq e < n \\ \mu_L & \text{if } e \geq n \end{cases}$$

In this model, P and Q could represent two processes running in the same operating system but on different CPUs. If the number of customers in P is greater or equal to n then we have a page swapping (i.e., slow persistent memory is used instead of fast volatile one). Thus, we have a decrease in the overall performance of the system. The service rate of the process represented by Q slows down and passes from μ_H (high) to μ_L (low) (e.g., it requires access to the same disk where the swapping is occurring). Q could decide to drop some of the requirements coming from the outside and hence the arrival rate of Q will pass from λ_H to λ_L, in order to maintain reasonable performance indices like expected response time.

P and Q can be in product-form if the following condition holds:

$$\frac{\lambda + \lambda_H}{\mu_H} = \frac{\lambda + \lambda_L}{\mu_L} = \rho_Q$$

In this case, we have that the stationary distribution for cooperation state (p, q) (of P and Q respectively) is:

$$\pi(p, q) = \pi_P(0)(1 - \rho_Q)\lambda^p \Big(\prod_{i=1}^{p} \frac{1}{\mu_p(i)} \Big) \rho_Q^q \tag{3.2}$$

A proof of this can be found in [8].

In literature, several studies have been made on the comparison of theoretical results for studying product-form models:

- [90]: quasi-reversible queueing networks are in product-form by ERCAT;
- [62]: derives the product-form for G-Networks with various types of synchronisations;
- Also Boucherie's product-form [18] has been proved via ERCAT.

It's not simple to prove that a model in product-form cannot be studied via ERCAT because we should prove that the conditions are violated for every possible assignment of the labels for the synchronisation specification. Moreover, the ERCAT analysis can be changed by the inversion of active and passive roles of unsynchronised labels, maintaining the same joint model. In this way, in the analysis the product-form can be identified in one case and not in the other. Even allowing the possible swap of active/passive transition roles, to avoid a computational cost higher than the direct solution of the global balance equation, in the system it is not allowed to change the label assigned to a specific transition.

Proposition 1. Given a pair of labelled models, ERCAT conditions are not necessary for product-form even considering possible swapping of the roles of the active/passive transitions.
The proof (found in [8]) relies on showing that the model does not satisfy the RCAT condition even if it's in product-form. Regardless of active/passive synchronisation roles,

the birth transitions have rate $\lambda_H(\lambda_L)$ and the death transitions $\mu_H(\mu_L)$. If the conditions of ERCAT theorem would be satisfied, we would have that:

$$\pi_P S'(s) \propto \Big(\frac{\lambda + \lambda_L + \lambda_H}{\mu_H + \mu_L}\Big)^s$$

which leads to a different expression of the joint stationary probabilities of Equation 3.2 for positive transition rates. Considering the uniqueness of stationary distribution and sufficiency of conditions of ERCAT, we have that ERCAT conditions cannot be satisfied. Even in the case that job completions in P are modelled by passive transitions and Q by active ones. The service rate would be different for each state of P and to make Q active we would need a denumerable number of labels in the synchronisation, leading us to a total birth rate of Q:

$$\lambda_L + \lambda_H + \sum_{p=1}^{\infty} \mu_P(e)$$

This would give us a different product-form. Using the same approach, we can prove that the product-form cannot be identified even if some transitions in P are made active and some others in S, to model the customer departures from P.

3.9 Conclusions

We saw in this chapter the compositional modelling. We gave a brief description of PEPA, Stochastic Automata and PIOAs. In the last part of this chapter we talked about product-forms, focusing on the presentation of the RCAT theorem and the notion of quasi reversibility. The definition of quasi-reversibility given by Kelly in [76] is also described using the notation of stochastic automata of Section 3.3 and PEPA 3.2. We presented also another way of define the RCAT theorem using the PEPA language.
The class of product-forms is wide and includes different kinds of systems. Even if the Reversed Compound Agent Theorem (RCAT) [61] and Extended Reversed Compound Agent Theorem (ERCAT) [63] are very general results for the analysis of product-form stochastic models, they have some limits and do not capture all product-forms models. There are some examples of meaningful models whose product-form cannot be analysed by applying these results. Consequently, finding other product-form systems and a more general theorem or method are still open problems. We will show in the next chapters some examples of product-forms whose stationary distribution cannot be derived via ERCAT or quasi-reversibility property.

Chapter 4

Propagation of Signals in Continuous Time

4.1 Introduction

In this chapter we will present propagation of signals in continuous time. We describe a case study with G-Networks with signals. In these networks, a customer is forced to move to another queue when a signal enters its queue and according to a Markovian routing rule it enters a new queue or leaves the network in batch mode. This is particularly useful to model synchronised or triggered motions; e.g., systems in which work and customers can be moved from one queue to another upon the arrival of an external or internal signal. Applications for G-Networks models with signals can be found in flow-control in communications systems. More complex systems can be analysed and developed with these models and in particular, they are outside the BCMP-network possibilities of solution presented in [14, 54]. We describe its product-form and stationary probability distribution. We also give a method to represent signals using the PEPA language. In order to do this we have to introduce an encoding method called Double Index (DI) solution, for modelling G-Networks with Triggers using the PEPA language. This method uses the concept of a double index. The double index in a process can trace fully the information about the state of another different process. This increases the "dependence" between the "tracker" process with respect to the "tracked" one, but completely eliminates any possible uncertainty about possible choices of tracker regarding actions of tracked processes. In this way a process can know exactly how many positive customers are in the system and where, and so it can decide to perform some specific kind of action. Furthermore, it can be also informed whether the departure of a customer will leave that queue emptied or not. We also use the notions of *Phantom State* and *Impossible Actions*. Impossible actions will never occur due to the fact that they will never cooperate with the corresponding active/passive actions in another process. In order to not cooperate with anyone, the so-called phantom state is added. This state will never be reachable because all the actions which lead to it are impossible actions. In this way all the actions from an unreachable state F will be impossible too, because they will never occur thanks to the non-reachability of F.

4.2 Case study: G-Networks with signals

We now consider queuing networks with customers and signals introduced by [50]. With a Markovian transition of a customer after its service, we can obtain both customers and signals in case they are not exogenous. A customer is forced to move to another queue when a signal enters its queue and according to a Markovian routing rule it enters the new queue or leaves the network in batch mode. This is particularly useful to model synchronised or triggered motions; e.g., systems in which work and customers can be moved from one queue to another upon the arrival of an external or internal signal. These networks are useful also for analyse system behaviours that we can observe in parallel computer system modelling and they have product-form solution and we can compute its customer routing equations [49].

4.2.1 Example

Let's consider an open queuing network with n servers mutually independent. Their service times have exponential rates $\mu_1, \mu_2, ..., \mu_n$. In the system, there are two types of entities which navigate through the network:

- Customers which add 1 to the queue length;
- Signals which trigger an instantaneous customer movement from one queue to another.

Both of them can arrive in i-th queue as an external arrival according to a Poisson process of rate λ_i for customers and γ_i for signals. The length of a queue is composed only by normal customers and their service is handled in the usual way. A customer after being served can:

- Leave the i-th queue and move to queue j with probability p_{ij}^+ as a normal customer;
- Leave the i-th queue and move to queue j with probability p_{ij}^-, becoming a signal;
- Depart from the system with probability $d_i = 1 - \sum_j (p_{ij}^+ + p_{ij}^-)$

The transition matrix of the underlying Markov chain is composed by elements $p_{ij} = p_{ij}^+ + p_{ij}^-$. This matrix represents the movements of both customers and signals.
Signals arriving in empty queues won't have any effect and just disappear from the system. On the contrary, if a queue is non-empty then one of the following events can occur:

- the signal arrives and triggers the movement of a customer from queue i to another queue j with probability q_{ij};
- the signal arrives and forces the departure from the system of a batch of customers with probability $D_i = 1 - \sum_j q_{ij}$.

Q is a $n \times n$ transition probability matrix. The length of the batch is B_i and its size distribution is general and given by $P[B_i = s] = L_{is}$ with $s \geq 1$. If the length of the queue is k_i at the moment of the signal arrival then:

- if $k_i \geq B_i$ then the length is reduced to $k_i - B_i$;
- otherwise the length is reduced to 0.

In this way a signal behaves as an external trigger which instantaneously moves a batch of customers to outside the system or a single customer from one queue to another. Assuming that p_{ij} and q_{ij} compose the transition probabilities of the transient of the Markov chain, we have that the probability that a customer leaves the system is guaranteed to be 1.

We define a n-vector of non-negative integers $k = (k_1, k_2, .., k_n)$ and the following (with $s \geq 1$):

- $k_i^{+s} = k_1, k_2, .., k_i + s, .., k_n$
- $k_i^{-} = k_1, k_2, .., k_i - 1, .., k_n$
- $k_{ij}^{+-} = k_1, k_2, .., k_i + 1, .., k_j - 1, .., k_n$
- $k_{ij}^{++s} = k_1, k_2, .., k_i + 1, .., k_j + s, .., k_n$
- $k_{ijm}^{++-} = k_1, k_2, .., k_i + 1, .., k_j + 1, .., k_m - 1, .., k_n$

$k(t) = (k_1(t), k_2(t), .., k_n(t))$ is the state of the system at time t and $k_i(t)$ is the number of customers in queue i at time t. $\{k(t) : t \geq 0\}$ represents the underlying continuous Markov chain of the state of the system and its steady-state distribution $\pi(k) \equiv \lim_{t\to\infty} P[k(t) = k]$ satisfies the following system of balance equation if its exists:

$$
\begin{aligned}
&\pi_k \sum_i [\lambda_i + (\gamma_i + \mu_i)\mathbf{1}[k_i > 0]] = \\
&= \sum_i \Bigg[\pi(k_i^+)\mu_i d_i + \pi(k_i^-)\lambda_i \mathbf{1}[k_1 > 0] + \lambda_i D_i \sum_{s=1}^{\infty} L_{is}\pi(k_i^{+s}) + \\
&+ \lambda_i D_i \sum_{s=1}^{\infty} L_{is} \sum_{u=0}^{s-1} \pi(k_i^{+u})\mathbf{1}[k_i = 0] + \\
&+ \sum_j \Big[\pi(k_{ij}^{+-})(\mu_i p_{ij}^+ + \gamma_i q_{ij})\mathbf{1}[k_j > 0] + \\
&+ \sum_{s=1}^{\infty} L_{is}\pi(k_{ij}^{++s})\mu_i p_{ij}^- D_j + \sum_{s=1}^{\infty} L_{is} \sum_{u=1}^{s-1} \pi(k_{ij}^{++u})\mu_i p_{ij}^- D_j \mathbf{1}[k_j = 0] + \\
&+ \pi(k_i^+)\mu_i p_{ij}^- \mathbf{1}[k_j = 0] + \\
&+ \sum_m \pi(k_{ijm}^{++-})\mu_i p_{ij}^- q_{jm}\mathbf{1}[k_m > 0] \Big] \Bigg]
\end{aligned}
\tag{4.1}
$$

where

$$
\mathbf{1}[x] = \begin{cases} 1 & \text{if } x \text{ is true} \\ 0 & \text{otherwise} \end{cases}
$$

The signal generation is handled in the last three terms of the equation:

- the first of them deals with the signal moving a batch of customers toward the outside of the system;
- the second represents a signal arriving in an empty queue and it just disappears from the system;

- the last covers the case of a signal arriving from queue i to queue j and instantaneously moving a customer from queue j to queue m (adding 1 customer to its length).

G-Networks were introduced in [48] and [52] in the single server queue case, under exponential assumption on service time and inter-arrival times. The signal models for G-networks were introduced in [50] and then generalize and further studied in [47] and [49], dealing with positive and negative customers. These signal models of [49] generalize the case of a single negative customer of [47, 48] which triggers an external departure of a single and positive customer in a queue (i.e., with probability $p_{ij}^{-}D_j$).

4.2.2 Special Case Application

We present an application for signal G-network models to control the flows in communications systems. More complex systems can be analysed and developed with these models and this is a special case of them but it's outside the BCMP-network possibilities of solution presented in [14, 54].
Let's consider a communication network composed by an input queue I and a sub-network S. S is composed by networks $N_1, N_2, .., N_s$ and provides alternate routes for the packet received by the network. These routes are disjointed. With some rate λ, packets enter the communication system from outside and wait in I. After that, they receive some service in I server, according to FIFO order and then are moved into some sub-network N_i. The packets can enter each sub-network N_i:

- With predetermined probability P_i to ensure a minimum of traffic flow in each sub-network;
- Under the effect of a signal due to a control flow decision. If in the sub-network N_i a packet leaves the system, it can accommodate another packet from I. This information, about its capacity, triggers the arrival of a new packet thanks to a control flow packet generated and sent back to input queue I. This control flow packet behaves as a signal and instantaneously moves a customer from I to N_i.

4.2.3 Product-Form

Considering the system of non-linear equations:

$$\begin{aligned}\gamma_i^{-} &= \sum_j \mu_j r_j p_{ji}^{-} + \gamma_i \\ \gamma_i^{+} &= \sum_j \mu_j r_j \left[p_{ji}^{+} + \sum_m p_{j,m}^{-} r_m r_{m,i}\right] + \sum_j \gamma_j r_j r_{ji} + \lambda_i\end{aligned} \tag{4.2}$$

where

$$r_i \equiv \gamma_i^{+}/[\mu_j + \gamma_i^{-} D_i f_i(r_i)]$$

$$f_i(x) = [1 - \sum_{s=1}^{\infty} L_i s x^s]/[1 - x]$$

and $i, j, m = 1, .., n$. We can rewrite γ_i^{+} as:

$$\gamma_i^{+} = \sum_j \mu_j r_j p_{ji}^{+} + \lambda_i + \sum_j \gamma_j^{-} r_j r_{ji}$$

Theorem 4.1. *If a non-negative solution* $\{\gamma_i^+, \gamma_i^-\}$ *exists to equation 4.2 such that each* $\gamma_i^+ < \mu_i + \gamma_i^- D_i f_i(r_i)$ *for* $i = 1, .., n$*, then*

$$\pi_k = \prod_{i=1}^{n} [1 - r_i] r_i^{ki}$$

The proof of this theorem for the case where only a single customer can be removed from the system (i.e., $P[B_i = 1] = 1$) can be found in [49].

4.2.4 Stability

The existence of the stationary probability distribution (i.e., the stability of the network) and the existence of a solution to the Equation 4.2 can be found extending the approach in [55] but considering also the effect of batch removals and signals which move packets from one queue to another one.

Theorem 4.2. *If the matrix* $[P^+ + Q]$ *is sub-stochastic and transient (i.e., it does not contain any ergodic class), then the solution of Equation 4.2,* $\{\gamma_i^+, \gamma_i^-\}$ *always exists for* $i = 1, .., n$*.*

The proof is based on Brouwer's theorem and can be found in [49]. The result about the existence of the product-form solution for this class of G-networks is found putting in the fixed-point vector $\mathbf{y}^*$ the values of γ_i^+ and γ_i^- and we can compute:

$$r_i(\mathbf{y}^*) = [\gamma_i^+ / \mu_i + \gamma_i^- D_i f_i(r_i)]\mathbf{y} = \mathbf{y}^*$$

for $i = 1, 2, .., n$.

Considering Theorem 4.1 and 4.2 the stationary solution $\pi_k = \lim_{t\to\infty} P[k(t) = k]$ exists if $r_i(\mathbf{y}^*) < 1$ and does not exist otherwise.

4.3 Representing signals in PEPA

In this section, we will first analyse an example of a type of G-Network with triggers. We briefly focus on its behaviour to form the basis of the subsequent model phase, searching for various aspects of the network:

- Dependences between components;
- Transitions of states in each component;
- Cooperations between components and their consequences
- Limits, particular cases and borderlines (like state 0 or state n)

After that we will model it with the PEPA language introduced in [70] using the Double Index (DI) method.
We will also compare this latter solution with the original system.

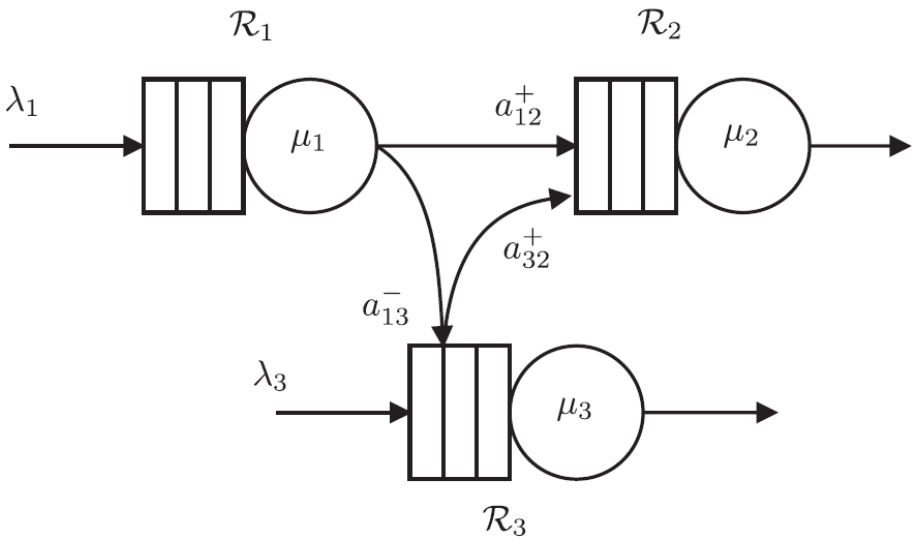


Figure 4.1: G-Network with trigger

4.3.1 G-Network with Trigger

The first example is a **G-Network with trigger** generated after job completion in R_1 that moves a customer from R_3 to R_2. It is represented by the Figure 4.1.
In this specific type of G-Network, a homogeneous Poisson process defines the arrival of positive customers, which arrive from outside the system to queue R_1 and R_3, with rates λ_1 and λ_3, respectively. Moreover, the rates μ_1, μ_2 and μ_3 represent the service times in each queue and so they are exponential random variables, which are independent. A customer of R_1, after the completion of its service can move to R_2 as a regular customer (a_{12}^+) or can travel to R_3 and change its nature into a trigger (a_{13}^-). In the second case, when R_3 is a non-empty queue, the length of its queue is reduced by one unit and a positive customer is added to the queue R_2 (a_{32}^+). If R_3 is empty when the trigger arrives, then nothing happen to R_3. We have to notice that when both R_1 and R_3 are non-empty queues, then the trigger causes the change of states in all three queues simultaneously.

If we consider the following informal annotations:

(r_1, r_2, r_3) is the state of the system, describing each queue by the number of its customers where:
r_1 is the number of positive customers in R_1,
r_2 is the number of positive customers in R_2,
r_3 is the number of positive customers in R_3.

And:

r_i^+ represents the addition of a positive customer to queue i (with $i = 1, 2, 3$) and is equivalent to $r_i^+ = r_i + 1$
r_i^- represents the completion of a service of a positive customer and its consequent departure from queue i (with $i = 1, 2, 3$) and is equivalent to $r_i^- = r_i - 1$

An overall informal description of this G-Network behaviour can be the following:

1) In R_1 a positive customer can arrive $\forall r_1 \geq 0$, this implies the change of overall state to (r_1^+, r_2, r_3)

2) In R_3 a positive customer can arrive $\forall r_3 \geq 0$, this implies the change of overall state to (r_1, r_2, r_3^+)

3) In R_1 a positive customer can have its service completed $\forall r_1 > 0$ and then it remains a positive customer and goes to R_2, this implies the change of overall state to (r_1^-, r_2^+, r_3)

4) In R_1 a positive customer can have its service completed $\forall r_1 > 0$ and then it becomes a trigger and goes to the empty queue R_3 (i.e. $r_3 = 0$) and it does nothing in R_3, this implies the change of overall state to $(r_1^-, r_2, 0)$

5) In R_1 a positive customer can have its service completed $\forall r_1 > 0$ and then it becomes a trigger and goes to the non-empty queue R_3 ($\forall r_3 > 0$) forcing a positive customer in the arrival queue to move in R_2 ($\forall r_2 \geq 0$), this implies the change of overall state to (r_1^-, r_2^+, r_3^-)

6) In R_2 a positive customer can have its service completed $\forall r_2 > 0$ and then it leaves the system as a positive customer, this implies the change of overall state to (r_1, r_2^-, r_3)

7) In R_3 a positive customer can have its service completed $\forall r_3 > 0$ and then it leaves the system as a positive customer, this implies the change of overall state to (r_1, r_2, r_3^-)

To summarize the arrivals and departures in each state:

- 1: arrival in R_1
- 3,4,5: departure in R_1
- 3,5: arrival in R_2
- 6: departure in R_2
- 2: arrival in R_3
- 5,7: departure in R_3

To summarize relations between state of components and actions:

- $r_1 \geq 0$ can do 1
- $r_1 > 0$ can do 1,3,4,5
- $r_2 \geq 0$ can do 3,5

- $r_2 > 0$ can do 3,5,6
- $r_3 \geq 0$ can do 2,5
- $r_3 > 0$ can do 2,5,7

And actions which change more queues are: 3 and 5

In the following section, we model the behaviour of this G-Network with the PEPA language.

4.3.1.1 Double Index (DI) Solution

We will now present the encoding of the G-Network with Trigger using DI method with the PEPA language. We adopt the annotation in which P_i is a generic state of R_1, Q_j of R_2 and R_k of R_3.

This solution uses the concept of a double index. The double index in R_1 traces fully the information about the state of R_3.
This increases the "dependence" of R_1 with respect to R_3 but completely eliminates any possible uncertainty of its possible choices regarding a_0 and a_1. R_1 knows exactly how many positive customers are in R_3 so it can decide to perform a_0 in case of empty R_3 or a_1 if R_3 has customers. Furthermore, R_1 knows also if the departure of the following customer in R_3 will leave R_3 empty or not.
Using the DI method to model a system, we have a growing in dependence of R_1 from R_3 and the increase of the state space, on the other side, there is a simplification in the complexity of actions and their correspondence with the behaviour of the original system and also in the whole encoding of the system.
Moreover, in a normal encoding the space of state for R_1 is n where n is the number of its customers; in this case, the space is $n * m$ where n is the number of customers of R_1 and m is the number of customers in R_3.
Thus R_1 is aware of all changes of R_3 and it is informed by R_3 itself, with the use of cooperation combinator in its actions: $fill$ increases its positive customers and $empty$ decreases them. In particular those types of actions help R_1 to keep track of the number of customers in R_3.
This information is stored in R_1 with the superscript number in the name of its state, thus:

- P_i represents the queue R_1 with i positive customers and the empty queue R_3 with 0 positive customers, $\forall i \geq 0$
- P_i^j represents the queue R_1 with i positive customers and the non-empty queue R_3 with j positive customers $\forall i \geq 0$ and $\forall j > 0$

The encoding of the G-Network in PEPA is the following:

$$R_1 \stackrel{def}{=} \begin{cases} P_0 = (\tau, \lambda_1).P_1 + (fill, \top).P_0^1 \\ P_0^m = (\tau, \lambda_1).P_1^m + (fill, \top).P_0^{m+1} + (empty, \top).P_0^{m-1} \\ P_n = (\tau, \lambda_1).P_{n+1} + (fill, \top).P_n' + (a, \mu_1 p).P_{n-1} + (a_0, \mu_1(1-p)).P_{n-1} \\ P_n^m = (\tau, \lambda_1).P_{n+1}^m + (fill, \top).P_n^{m+1} + (empty, \top).P_n^{m-1} + (a, \mu_1 p).P_{n-1}^m + \\ \qquad +(a_1, \mu_1(1-p)).P_{n-1}^{m-1} \end{cases}$$

$$R_2 \stackrel{def}{=} \begin{cases} Q_0 = (a, \top).Q_1 + (a_1, \top).Q_1 \\ Q_n = (a, \top).Q_{n+1} + (a_1, \top).Q_{n+1} + (\tau, \mu_2).Q_{n-1} \end{cases}$$

$$R_3 \stackrel{def}{=} \begin{cases} V_0 = (fill, \lambda_3).V_1 + (a_0, \top).V_0 \\ V_n = (fill, \lambda_3).V_{n+1} + (empty, \mu_3).V_{n-1} + (a_1, \top).V_{n-1} \qquad n > 0 \end{cases}$$

With $m > 0$ and $n > 0$ and considering $P_n^0 \equiv P_n$ for better readability.

With this representation, we can define the following correspondence between informal description and encoding:

- 1: represented by (τ, λ_1) in R_1
- 2: represented by $(fill, \lambda_3)$ in R_3 (and R_1 is informed by $(fill, \top)$)
- 3: represented by $(a, \mu_1 p)$ in R_1 and by $(a, \top)$ in R_2
- 4: represented by $(a_0, \mu_1(1-p))$ in R_1 and by $(a_0, \top)$ in R_3
- 5: represented by $(a_1, \mu_1(1-p))$ in R_1 and by $(a_1, \top)$ in both R_2 and R_3
- 6: represented by (τ, μ_2) in R_2
- 7: represented by $(empty, \lambda_3)$ in R_3 (and R_1 is informed with $(empty, \top)$)

4.3.2 G-Network Iterative Customer Removals

In this second example a **G-etwork with iterative customer removals** started with the arrival of an external negative trigger β, this system adds also the concept of *chains* of actions in the model, further than the previous example. This concatenation of a lot of instant actions will stress my solution and enlighten its advantages and its weaknesses.
It is represented by the Figure 4.2.
In this example there is the use of a negative trigger β. Negative and positive triggers are very similar, except that a negative one transforms a removed customer into a trigger itself (which can be positive or negative). Thus, if a negative trigger is generated from a removed customer, its arrival to the next queue will be also a trigger and this can lead to the formation of chains of negative triggers in a system. Those triggers will stop only if they move to an empty queue or they become a positive trigger or leave the network (like ordinary negative customers) and have a strictly specific order of their routing (i.e., trigger with route $1, 2, 3$ has different behaviour from one with route $1, 3, 2$). In this way a whole subset of queues of the system can be emptied.
In this specific type of G-Network, a homogeneous Poisson process defines the arrival of

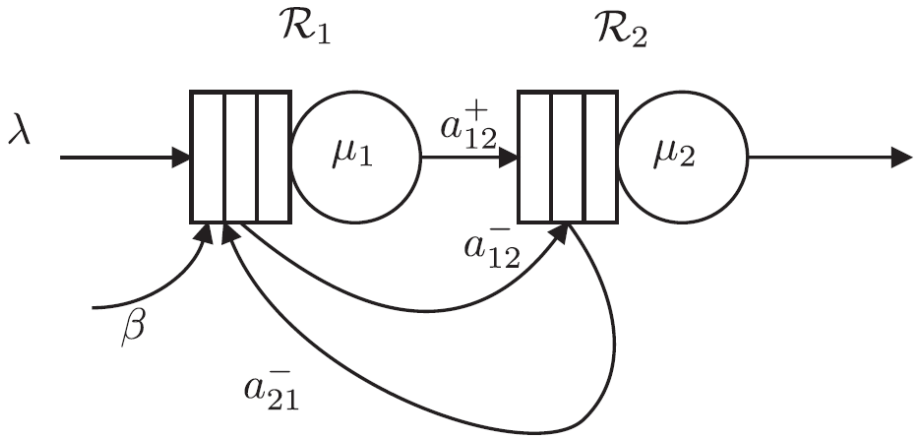


Figure 4.2: G-Network with iterative customer removals

positive customers, which arrive from outside the system to queue R_1, with rate λ. Moreover, they are served first in R_1 and then in R_2, the rates μ_1 and μ_2 represent in fact the service times in each queue and so they are exponential random variables, which are independent. A customer of R_1, after the completion of its service, moves to R_2 as a regular customer (a_{12}^+). A negative trigger can arrive with rate β in R_1 and it reduces the length of its queue by one unit if R_1 is a non-empty queue. Moreover, that removed customer become a negative trigger and it will instantaneously propagate to R_2, then back to R_1, then to R_2 again and so on, until one of the two queues is emptied when the chain of triggers arrives. In this way the chain terminates.
(a_{12}^-) represents the propagation of the trigger from R_1 to R_2 and (a_{21}^-) represents the opposite. Thus, this example allows the construction of chains of instantaneous transitions between the two queues. These chains are finite even if they are also unbounded in length. Obviously, we have to notice that when both R_1 and R_2 are non-empty queues, then the trigger causes the change of states in both of them simultaneously and usually with a high change in their numbers of positive customers.

If we consider the following informal annotations:

(r_1, r_2) is the state of the system, describing each queue with the number of its customers where:
r_1 is the number of positive customers in R_1,
r_2 is the number of positive customers in R_2,

And:

r_i^+ represents the addition of a positive customer to queue i (with $i = 1, 2$) and is equivalent to $r_i^+ = r_i + 1$
r_i^- represents the completion of a service of a positive customer and its consequently departure from queue i (with $i = 1, 2$) and is equivalent to $r_i^- = r_i - 1$

An overall informal description of this G-Network behaviour can be the following:

1) In R_1 a positive customer can arrive $\forall r_1 \geq 0$, this implies the change of overall state to (r_1^+, r_2)

2) In R_1 a positive customer can have its service completed $\forall r_1 > 0$ and then it remains a positive customer and goes to R_2, this implies the change of overall state to (r_1^-, r_2^+)

3) In R_2 a positive customer can have its service completed $\forall r_2 > 0$ and then it leaves the system as a positive customer, this implies the change of overall state to (r_1, r_2^-)

4) In R_1 when there is an empty queue, a negative trigger can arrive from either outside the system or R_2 and it stops itself and does nothing in R_1, the overall state remains $(0, r_2)$

5) In R_1 a negative trigger can arrive from either outside the system or R_2 $\forall r_1 > 0$ and then, after it kills a positive customer in the queue, it propagates immediately to R_2, this implies the change of overall state to (r_1^-, r_2)

6) In R_2 when there is an empty queue, a negative trigger can arrive from R_1 and it stops itself and does nothing in R_2, this doesn't change the overall state $(r_1, 0)$

7) In R_2 a negative trigger can arrive from R_1 $\forall r_2 > 0$ and then, after it kills a positive customer in the queue, it propagates immediately to R_1, this implies the change of overall state to (r_1, r_2^-)

To summarize the arrivals and departures in each state:

- 1: arrival in R_1
- 2,5: departure in R_1
- 2: arrival in R_2
- 3,7: departure in R_2

To summarize relations between state of components and actions:

- $r_1 \geq 0$ can do 1,4
- $r_1 > 0$ can do 1,2,5
- $r_2 \geq 0$ can do 2,6
- $r_2 > 0$ can do 2,3,7

And the only action which changes more queues is 2, but 5 and 7 are part of a chain of changes.

In the following section, we model the behaviour of this G-Network with the PEPA language.

4.3.2.1 Double Index (DI) Solution

We will now present the stress test of DI solution with the encoding of the G-Network with Iterative Customer Removals using PEPA. We adopt the annotation in which P_i is a generic state of R_1 and Q_j of R_2.

In this trial we use again the concept of double index but with a further type of representation. In this encoding we use the concept of $MAX(i, j)$ because there are two main cases in both queues.
The two cases are:

- the number of customers of R_1 is less than the number of R_2; in this case the negative trigger will empty the R_1 queue;
- the number of customers of R_1 is more than the number of R_2; in this case the negative trigger will subtract only a certain number of customers in R_1 queue.

It is more or less the same, also in the case of the Q process.
This is done in order to avoid negative results from subtractions because negative states don't exist in this system. Moreover, this represents only a single choice because in each instantiation of the model $MAX(i-n; 0)$ and $MAX(n-i-1; 0)$ are gradually substituted by the result of the MAX operator (which behaves as the normal operator of maximum in mathematics) and usually they are complementary (i.e., if the first possibility is chosen in the first MAX then in the second is chosen the second one, and vice versa).
In this case the double indexes in both R_1 and R_2 mutually trace fully the information about the states of R_2 and R_1.
We make this encoding because both R_1 and R_2 need to know the state of the other to take a unique and right decision about the outcome of the negative trigger effect. R_1 knows exactly how many positive customers are in R_2 so it can decide how many positive customers it removes from its queue when a trigger arrives and R_2 behaves in the same way but with respect to R_1. Thus, we retain the simplification in the complexity of actions and their correspondence with the behaviour of the original system and also in the whole encoding of the system.
Then, similarly to previous encodings, R_1 is aware of all changes of R_2 and it is informed by R_2 itself, with the use of cooperation combinator in its action a_2 which decreases the positive customers in R_2 and similarly R_2 is informed by R_1 itself with the use of cooperation combinator in its action a_0 which increases its positive customers. In particular those types of actions help R_1 and R_2 to mutually keep track of their number of customers.
This information is stored in R_1 and R_2 with the superscript number in the name of their state, thus:

- P_i represents the queue R_1 with i positive customers and the empty queue R_2 with 0 positive customers, $\forall i \geq 0$
- P_i^j represents the queue R_1 with i positive customers and the non-empty queue R_2 with j positive customers $\forall i \geq 0$ and $\forall j > 0$

And vice versa for Q in R_2.

The encoding of the G-Network in PEPA is the following:

$$R_1 = \begin{cases} P_0 = (a_0, \lambda).P_1 + (\tau, \beta).P_0 \\ P_n = (a_0, \lambda).P_{n+1} + (a_1, \mu_1).P^1_{n-1} + (\tau, \beta).P_{n-1} \\ P^i_0 = (a_0, \lambda).P^i_1 + (a_2, \top).P^{i-1}_0 + (\tau, \beta).P^i_0 \\ P^i_n = (a_0, \lambda).P^i_{n+1} + (a_1, \mu_1).P^{i+1}_{n-1} + (a_2, \top).P^{i-1}_n + (b, \beta).P^{MAX(i-n;0)}_{MAX(n-i-1;0)} \end{cases}$$

$$R_2 = \begin{cases} Q_0 = (a_0, \top).Q^1_0 \\ Q_n = (a_0, \top).Q^1_n + (a_2, \mu_2).Q_{n-1} \\ Q^i_0 = (a_0, \top).Q^{i+1}_0 + (a_1, \top).Q^{i-1}_1 \\ Q^i_0 = (a_0, \top).Q^{i+1}_n + (a_1, \top).Q^{i-1}_{n+1} + (a_2, \mu_2).Q^i_{n-1} + (b, \top).Q^{MAX(i-n-1;0)}_{MAX(n-i;0)} \end{cases}$$

With $n > 0$ and $i > 0$ and considering $P^0_n \equiv P_n$ and $Q^0_n \equiv Q_n$ for better readability.

With this representation, we can define the following correspondence between informal description and encoding:

- 1: represented by (a_0, λ) in R_1 (and R_2 is informed with $(a_0, \top)$)
- 2: represented by (a_1, μ_1) in R_1 and by $(a_1, \top)$ in R_2
- 3: represented by (a_2, μ_2) in R_2 (and R_1 is informed with $(a_2, \top)$)
- 4: represented by (τ, β) in R_1 or the conclusion of the subtraction caused by (b, β) in R_1
- 5: represented by the subtraction caused by (b, β) in R_1
- 6: represented by the conclusion of the subtraction caused by $(b, \top)$, in R_2
- 7: represented by the subtraction caused by $(b, \top)$ in R_2

4.3.3 Derivation Graphs of G-Network with Trigger

In this part we draw the derivation Graphs of *G-Network with Trigger* 4.1, to see visually its behaviours in Figure 4.3.
As expected, the graph grows in two dimensions: horizontal and vertical. The various columns represent the number of normal customers in R_1 instead the rows represent the information stored in R_1 about the number of positive customers in R_3. This means that the state $2'$ represents R_1 with 2 positive customers and R_3 with only 1 positive customer.

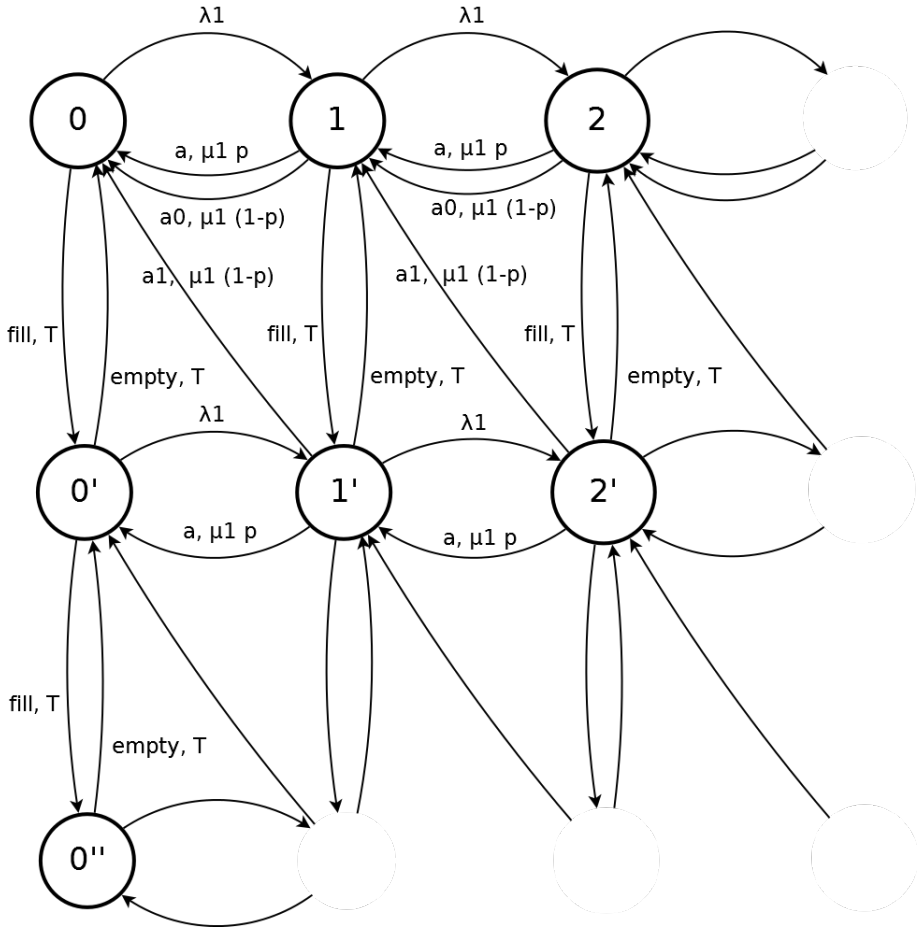


Figure 4.3: Derivation Graph of process R_1

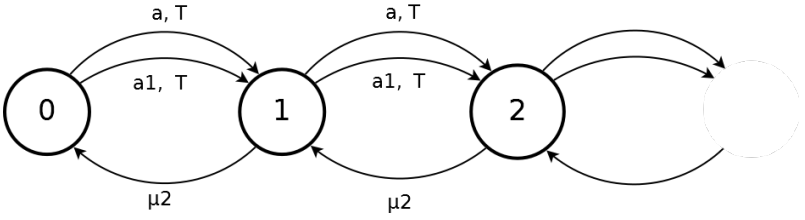


Figure 4.4: Derivation Graph of process R_2

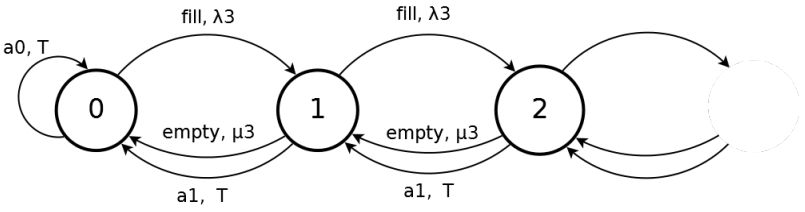


Figure 4.5: Derivation Graph of process R_3

4.3.4 Comparison between two systems

Now we will show that DI system in PEPA has the same behaviour to the original one shown in Figure 4.1.

Hypothesis:
1) The G-Network will be represented by ($\overline{n_1}$, $\overline{n_2}$, $\overline{n_3}$), namely the states of its queues, where:

- $\overline{n_1}$: represents the number of positive customers in queue R_1 of the G-Network
- $\overline{n_2}$: represents the number of positive customers in queue R_2 of the G-Network
- $\overline{n_3}$: represents the number of positive customers in queue R_3 of the G-Network

So the original system will be $(\overline{n_1}, \overline{n_2}, \overline{n_3})$

2) My system will also be represented by ((n_{11}, n_{13}), n_2, n_3), namely the states of my processes P (the first two numbers), Q and R, where:

- n_{11}: represents the number i in the state name of a general process P_i^j, i.e., the number of positive customers in process R_1 of my encoding
- n_{13}: represents the number j in the superscript name of a general process P_i^j, i.e., the information about the number of positive customers in process R_3 of my encoding, stored in R_1
- n_2: represents the number i in the state name of a general process Q_i, i.e., the number of positive customers in process R_2 of my encoding
- n_3: represents the number i in the state name of a general process R_i, i.e., the number of positive customers in process R_3 of my encoding

So my model will be $S = ((n_{11}, n_{13}), n_2, n_3)$
It is important to note that (n_{11}, n_{13}) is the state of process R_1, n_2 of R_2, n_3 of R_3. Moreover, it is important to notice that since they model the same number of customers:

- $\overline{n_1} = n_{11}$
- $\overline{n_2} = n_2$
- $\overline{n_3} = n_3$
- $n_{13} = n_3$.

My system will be represented as: $S \stackrel{def}{=} P \underset{a,a_0,a_1,fill,empty}{\bowtie} Q \underset{a,a_0,a_1,fill,empty}{\bowtie} V$

Thesis:
The DI model represents the systems depicted in Figure 4.1. We will see that we have an isomorphism of the labelled transition system of the CTMC underlying the original system with our model.

4.3.4.1 Possible Actions of two Systems

The original system $(\overline{n_1}, \overline{n_2}, \overline{n_3})$ can perform the following actions (i.e., transition between states), with also the following restrictions and rates:

1. Informal description: A positive customer arrives to queue R_1
   Action 1: from $(\overline{n_1}, \overline{n_2}, \overline{n_3})$ to $(\overline{n_1 + 1}, \overline{n_2}, \overline{n_3})$
   Restriction: $\forall \overline{n_1} \geq 0$
   Rate: λ_1

2. Informal description: A positive customer arrives to queue R_3
   Action 2: from $(\overline{n_1}, \overline{n_2}, \overline{n_3})$ to $(\overline{n_1}, \overline{n_2}, \overline{n_3 + 1})$
   Restriction: $\forall \overline{n_3} \geq 0$
   Rate: λ_3

3. Informal description: A positive customer moves from queue R_1 to queue R_2
   Action 3: from $(\overline{n_1}, \overline{n_2}, \overline{n_3})$ to $(\overline{n_1 - 1}, \overline{n_2 + 1}, \overline{n_3})$
   Restriction: $\forall \overline{n_1} \geq 1$ and $\forall \overline{n_2} \geq 0$
   Rate: $\mu_1 p$

4. Informal description: A positive customer moves from queue R_1 becomes a trigger and arrives to queue R_3 when it is empty
   Action 4: from $(\overline{n_1}, \overline{n_2}, 0)$ to $(\overline{n_1 - 1}, \overline{n_2}, 0)$
   Restriction: $\forall \overline{n_1} \geq 1$ and $\overline{n_3} = 0$
   Rate: $\mu_1(1 - p)$

5. Informal description: A positive customer moves from queue R_1 becomes a trigger and arrives to queue R_3 when it is non-empty, then a positive customer from R_3 is forced to move to R_2
   Action 5: from $(\overline{n_1}, \overline{n_2}, \overline{n_3})$ to $(\overline{n_1 - 1}, \overline{n_2 + 1}, \overline{n_3 - 1})$
   Restriction: $\forall \overline{n_1} \geq 1$ and $\forall \overline{n_2} \geq 0$ and $\forall \overline{n_3} \geq 1$
   Rate: $\mu_1(1 - p)$

6. Informal description: A positive customer leaves the system from queue R_2
   Action 6: from $(\overline{n_1}, \overline{n_2}, \overline{n_3})$ to $(\overline{n_1}, \overline{n_2 - 1}, \overline{n_3})$
   Restriction: $\forall \overline{n_2} \geq 1$
   Rate: μ_2

7. Informal description: A positive customer leaves the system from queue R_3
   Action 7: from $(\overline{n_1}, \overline{n_2}, \overline{n_3})$ to $(\overline{n_1}, \overline{n_2}, \overline{n_3 - 1})$
   Restriction: $\forall \overline{n_3} \geq 1$
   Rate: μ_3

We will now define an update vector for each of these actions in which we can track the differences in the numbers of customers in each queue:

1. Action 1: from $(\overline{n_1}, \overline{n_2}, \overline{n_3})$ to $(\overline{n_1 + 1}, \overline{n_2}, \overline{n_3})$
   $(1, \lambda_1)$: $(+1, 0, 0)$

2. Action 2: from $(\overline{n_1}, \overline{n_2}, \overline{n_3})$ to $(\overline{n_1}, \overline{n_2}, \overline{n_3 + 1})$
   $(2, \lambda_3)$: $(0, 0, +1)$

3. Action 3: from $(\overline{n_1}, \overline{n_2}, \overline{n_3})$ to $(\overline{n_1 - 1}, \overline{n_2 + 1}, \overline{n_3})$
   $(3, \mu_1 p)$: $(-1, +1, 0)$
4. Action 4: from $(\overline{n_1}, \overline{n_2}, 0)$ to $(\overline{n_1 - 1}, \overline{n_2}, 0)$
   $(4, \mu_1(1-p))$: $(-1, 0, 0)$
5. Action 5: from $(\overline{n_1}, \overline{n_2}, \overline{n_3})$ to $(\overline{n_1 - 1}, \overline{n_2 + 1}, \overline{n_3 - 1})$
   $(5, \mu_1(1-p))$: $(-1, +1, -1)$
6. Action 6: from $(\overline{n_1}, \overline{n_2}, \overline{n_3})$ to $(\overline{n_1}, \overline{n_2 - 1}, \overline{n_3})$
   $(6, \mu_2)$: $(0, -1, 0)$
7. Action 7: from $(\overline{n_1}, \overline{n_2}, \overline{n_3})$ to $(\overline{n_1}, \overline{n_2}, \overline{n_3 - 1})$
   $(7, \mu_3)$: $(0, 0, -1)$

From the derivation graphs, we can analyse each process of the model, their actions (i.e., transition between states), with also their restrictions and rates.
Process P which represents R_1 can perform:

- State P_0
  - Action: τ
    Restriction: none
    Rate: λ_1
  - Action: $fill$
    Restriction: cooperate, has to wait for R_3 action of $fill$
    Rate: $\top$
- State P_n with $n > 0$
  - Action: τ
    Restriction: none
    Rate: λ_1
  - Action: $fill$
    Restriction: cooperate, has to wait for R_3 action of $fill$
    Rate: $\top$
  - Action: a
    Restriction: none, but there is an active cooperation with R_2
    Rate: $\mu_1 p$
  - Action: a_0
    Restriction: none, but there is an active cooperation with R_3, moreover R_3 must be empty but this is ensured by the fact that there wasn't any $fill$ action in R_3 otherwise the state will be P_n^i with $i > 0$
    Rate: $\mu_1(1-p)$
- State P_0^i with $i > 0$
  - Action: τ
    Restriction: none
    Rate: λ_1

  - Action: $fill$
    Restriction: cooperate, has to wait for R_3 action of $fill$
    Rate: $\top$
  - Action: $empty$
    Restriction: cooperate, has to wait for R_3 action of $empty$
    Rate: $\top$

- State P_n^i with $n > 0$ and $i > 0$
  - Action: τ
    Restriction: none
    Rate: λ_1
  - Action: $fill$
    Restriction: cooperate, has to wait for R_3 action of $fill$
    Rate: $\top$
  - Action: $empty$
    Restriction: cooperate, has to wait for R_3 action of $empty$
    Rate: $\top$
  - Action: a
    Restriction: none, but there is an active cooperation with R_2
    Rate: $\mu_1 p$
  - Action: a_1
    Restriction: none, but there is an active cooperation with R_3, moreover R_3 must be non-empty but this is ensured by the fact that there were one or more $fill$ action in R_3 otherwise the state will be only P_n with $i = 0$
    Rate: $\mu_1(1-p)$

Process Q which represents R_2 can perform:

- State Q_0
  - Action: a
    Restriction: cooperate, has to wait for R_1 action of a
    Rate: $\top$
  - Action: a_1
    Restriction: cooperate, has to wait for R_1 action of a_1 (so R_1 and R_3 must be non-empty queues)
    Rate: $\top$

- State Q_n with $n > 0$
  - Action: a
    Restriction: cooperate, has to wait for R_1 action of a
    Rate: $\top$
  - Action: a_1
    Restriction: cooperate, has to wait for R_1 action of a_1 (so R_1 and R_3 must be non-empty queues)
    Rate: $\top$

  - Action: τ
    Restriction: none
    Rate: μ_2

Process V which represents R_3 can perform:

- State V_0
  - Action: $fill$
    Restriction: none, but there is an active cooperation with R_1
    Rate: λ_3
  - Action: a_0
    Restriction: cooperate, has to wait for R_1 action of a_0
    Rate: $\top$
- State V_n with $n > 0$
  - Action: $fill$
    Restriction: none, but there is an active cooperation with R_1
    Rate: λ_3
  - Action: $empty$
    Restriction: none, but there is an active cooperation with R_1
    Rate: μ_3
  - Action: a_1
    Restriction: cooperate, has to wait for R_1 action of a_1
    Rate: $\top$

Where there are no further specifications, when a process waits for a cooperation (i.e., it is passive with respect to that action) means that the conditions for that activity are imposed only by the active process. Moreover, when a process doesn't have any restriction in an activity but there is an active cooperation, it means that there are no conditions in that activity (except the ones in its state, e.g., if the activity is present only in P_n and not in P_0 this means that one condition on that activity is $n > 0$)

To summarize, the model encoded from original system $((n_{11}, n_{13}), n_2, n_3)$ can perform the following actions (i.e., transition between states), with also the following restrictions and rates:

1. Action τ: from $((n_{11}, n_{13}), n_2, n_3)$ to $((n_{11} + 1, n_{13}), n_2, n_3)$
   Restriction: $\forall n_{11} \geq 0$
   Rate: λ_1

2. Action $fill$: from $((n_{11}, n_{13}), n_2, n_3)$ to $((n_{11}, n_{13} + 1), n_2, n_3 + 1)$
   Restriction: $\forall n_{13} = n_3 \geq 0$
   Rate: λ_3

3. Action a: from $((n_{11}, n_{13}), n_2, n_3)$ to $((n_{11} - 1, n_{13}), n_2 + 1, n_3)$
   Restriction:$\forall n_{11} \geq 1$ and $\forall n_2 \geq 0$
   Rate: $\mu_1 p$

4. Action a_0: from $((n_{11}, 0), n_2, 0)$ to $((n_{11} - 1, 0), n_2, 0)$
   Restriction:$\forall n_{11} \geq 1$ and $\forall n_{13} = n_3 = 0$
   Rate: $\mu_1(1 - p)$

5. Action a_1: from $((n_{11}, n_{13}), n_2, n_3)$ to $((n_{11} - 1, n_{13} - 1), n_2 + 1, n_3 - 1)$
   Restriction:$\forall n_{11} \geq 1$ and $\forall n_{13} = n_3 \geq 1$ and $\forall n_2 \geq 0$
   Rate: $\mu_1(1 - p)$

6. Action τ: from $((n_{11}, n_{13}), n_2, n_3)$ to $((n_{11}, n_{13}), n_2 - 1, n_3)$
   Restriction: $\forall n_2 \geq 1$
   Rate: μ_2

7. Action $empty$: from $((n_{11}, n_{13}), n_2, n_3)$ to $((n_{11}, n_{13} - 1), n_2 - 1, n_3 - 1)$
   Restriction: $\forall n_{13} = n_3 \geq 1$
   Rate: μ_3

We will now define an update vector for each of these actions in which we can track the differences in the numbers of customers in each queue:

1. Action τ: from $((n_{11}, n_{13}), n_2, n_3)$ to $((n_{11} + 1, n_{13}), n_2, n_3)$
   (τ, λ_1): $((+1, 0), 0, 0)$

2. Action $fill$: from $((n_{11}, n_{13}), n_2, n_3)$ to $((n_{11}, n_{13} + 1), n_2, n_3 + 1)$
   $(fill, \lambda_3)$: $((0, +1), 0, +1)$

3. Action a: from $((n_{11}, n_{13}), n_2, n_3)$ to $((n_{11} - 1, n_{13}), n_2 + 1, n_3)$
   $(a, \mu_1 p)$: $((-1, 0), +1, 0)$

4. Action a_0: from $((n_{11}, 0), n_2, 0)$ to $((n_{11} - 1, 0), n_2, 0)$
   $(a_0, \mu_1(1 - p))$: $((-1, 0), 0, 0)$

5. Action a_1: from $((n_{11}, n_{13}), n_2, n_3)$ to $((n_{11} - 1, n_{13} - 1), n_2 + 1, n_3 - 1)$
   $(a_1, \mu_1(1 - p))$: $((-1, -1), +1, -1)$

6. Action τ: from $((n_{11}, n_{13}), n_2, n_3)$ to $((n_{11}, n_{13}), n_2 - 1, n_3)$
   (τ, μ_2): $((0, 0), -1, 0)$

7. Action $empty$: from $((n_{11}, n_{13}), n_2, n_3)$ to $((n_{11}, n_{13} - 1), n_2 - 1, n_3 - 1)$
   $(empty, \mu_3)$: $((0, -1), 0, -1)$

We can compare now the update vectors from the original system and from the DI method:

1. Action 1 has a similar behaviour to Action τ:
   from $(\overline{n_1}, \overline{n_2}, \overline{n_3})$ to $(\overline{n_1 + 1}, \overline{n_2}, \overline{n_3})$
   from $((n_{11}, n_{13}), n_2, n_3)$ to $((n_{11} + 1, n_{13}), n_2, n_3)$
   $(1, \lambda_1)$: $(+1, 0, 0)$
   $((+1, 0), 0, 0)$

2. Action 2 has a similar behaviour to Action $fill$:
   from $(\overline{n_1}, \overline{n_2}, \overline{n_3})$ to $(\overline{n_1}, \overline{n_2}, \overline{n_3 + 1})$
   from $((n_{11}, n_{13}), n_2, n_3)$ to $((n_{11}, n_{13} + 1), n_2, n_3 + 1)$
   $(2, \lambda_3)$: $(0, 0, +1)$
   $(fill, \lambda_3)$: $((0, +1), 0, +1)$

3. Action 3 has a similar behaviour to Action a:
   from $(\overline{n_1}, \overline{n_2}, \overline{n_3})$ to $(\overline{n_1 - 1}, \overline{n_2 + 1}, \overline{n_3})$
   from $((n_{11}, n_{13}), n_2, n_3)$ to $((n_{11} - 1, n_{13}), n_2 + 1, n_3)$
   $(3, \mu_1 p)$: $(-1, +1, 0)$
   $(a, \mu_1 p)$: $((-1, 0), +1, 0)$

4. Action 4 has a similar behaviour to Action a_0:
   from $(\overline{n_1}, \overline{n_2}, 0)$ to $(\overline{n_1 - 1}, \overline{n_2}, 0)$
   from $((n_{11}, 0), n_2, 0)$ to $((n_{11} - 1, 0), n_2, 0)$
   $(4, \mu_1(1 - p))$: $(-1, 0, 0)$
   $(a_0, \mu_1(1 - p))$: $((-1, 0), 0, 0)$

5. Action 5 has a similar behaviour to Action a_1:
   from $(\overline{n_1}, \overline{n_2}, \overline{n_3})$ to $(\overline{n_1 - 1}, \overline{n_2 + 1}, \overline{n_3 - 1})$
   from $((n_{11}, n_{13}), n_2, n_3)$ to $((n_{11} - 1, n_{13} - 1), n_2 + 1, n_3 - 1)$
   $(5, \mu_1(1 - p))$: $(-1, +1, -1)$
   $(a_1, \mu_1(1 - p))$: $((-1, -1), +1, -1)$

6. Action 6 has a similar behaviour to Action τ:
   from $(\overline{n_1}, \overline{n_2}, \overline{n_3})$ to $(\overline{n_1}, \overline{n_2 - 1}, \overline{n_3})$
   from $((n_{11}, n_{13}), n_2, n_3)$ to $((n_{11}, n_{13}), n_2 - 1, n_3)$
   $(6, \mu_2)$: $(0, -1, 0)$
   (τ, μ_2): $((0, 0), -1, 0)$

7. Action 7 has a similar behaviour to Action $empty$:
   from $(\overline{n_1}, \overline{n_2}, \overline{n_3})$ to $(\overline{n_1}, \overline{n_2}, \overline{n_3 - 1})$
   from $((n_{11}, n_{13}), n_2, n_3)$ to $((n_{11}, n_{13} - 1), n_2 - 1, n_3 - 1)$
   $(7, \mu_3)$: $(0, 0, -1)$
   $(empty, \mu_3)$: $((0, -1), 0, -1)$

The update vectors of both systems affect them in the same way, i.e. in both the change of number of customers and where they can occur. To better understand this, we will show that they are applied in the same states of the systems. We have the following cases in the original system:

1. $(\overline{0}, \overline{0}, \overline{0})$: all queues are empty.

2. $(\overline{n}, \overline{0}, \overline{0})$: R_1 has n positive customers, R_2 and R_3 are empty.

3. $(\overline{0}, \overline{n}, \overline{0})$: R_2 has n positive customers, R_1 and R_3 are empty.

4. $(\overline{0}, \overline{0}, \overline{n})$: R_3 has n positive customers, R_1 and R_2 are empty.

5. $(\overline{n_1}, \overline{n_2}, \overline{0})$: R_1 has n_1 positive customers, R_2 has n_2 positive customers and R_3 is empty.

6. $(\overline{n_1}, \overline{0}, \overline{n_3})$: R_1 has n_1 positive customers, R_3 has n_3 positive customers and R_2 is empty.

7. $(\overline{0}, \overline{n_2}, \overline{n_3})$: R_2 has n_2 positive customers, R_3 has n_3 positive customers and R_1 is empty.

8. $(\overline{n_1}, \overline{n_2}, \overline{n_3})$: R_1 has n_1 positive customers, R_2 has n_2 positive customers, R_3 has n_3 positive customers.

Case 1
From state:
$(\overline{0}, \overline{0}, \overline{0})$

We can have only two possible actions with these values: Action 1 and Action 2.
And these are the two possible state transitions:

1. $(\overline{0}, \overline{0}, \overline{0}) \xrightarrow{1,\lambda_1} (\overline{1}, \overline{0}, \overline{0})$

2. $(\overline{0}, \overline{0}, \overline{0}) \xrightarrow{2,\lambda_3} (\overline{0}, \overline{0}, \overline{1})$

There exists the following correspondence in my system $((0,0),0,0)$ with actions: τ and $fill$ which have the same update vectors of Action 1 and Action 2. And these are the two possible state transitions:

1. $((0,0),0,0) \xrightarrow{\tau,\lambda_1} ((1,0),0,0)$

2. $((0,0),0,0) \xrightarrow{fill,\lambda_3} ((0,1),0,1)$

Case 2
From state:
$(\overline{n}, \overline{0}, \overline{0})$

We can have only four possible actions with these values: Action 1, Action 2, Action 3 and Action 4.
And these are the four possible state transitions:

1. $(\overline{n}, \overline{0}, \overline{0}) \xrightarrow{1,\lambda_1} (\overline{(n+1)}, \overline{0}, \overline{0})$

2. $(\overline{n}, \overline{0}, \overline{0}) \xrightarrow{2,\lambda_3} (\overline{n}, \overline{0}, \overline{1})$

3. $(\overline{n}, \overline{0}, \overline{0}) \xrightarrow{3,\mu_1 p} (\overline{(n-1)}, \overline{1}, \overline{0})$

4. $(\overline{n}, \overline{0}, \overline{0}) \xrightarrow{4,\mu_1(1-p)} (\overline{(n-1)}, \overline{0}, \overline{0})$

There exists the following correspondence in my system $((n,0),0,0)$ with actions: τ, $fill$, a, a_0 which have the same update vectors. And these are the four possible state transitions:

1. $((n,0),0,0) \xrightarrow{\tau,\lambda_1} ((n+1,0),0,0)$

2. $((n,0),0,0) \xrightarrow{fill,\lambda_3} ((n,1),0,1)$

3. $((n,0),0,0) \xrightarrow{a,\mu_1 p} ((n-1,0),1,0)$

4. $((n,0),0,0) \xrightarrow{a_0,\mu_1(1-p)} ((n-1,0),0,0)$

Case 3
From state:
$(\overline{0},\overline{n},\overline{0})$

We can have only three possible actions with these values: Action 1, Action 2 and Action 6.
And these are the three possible state transitions:

1. $(\overline{0},\overline{n},\overline{0}) \xrightarrow{1,\lambda_1} (\overline{1},\overline{n},\overline{0})$

2. $(\overline{0},\overline{n},\overline{0}) \xrightarrow{2,\lambda_3} (\overline{0},\overline{n},\overline{1})$

3. $(\overline{0},\overline{n},\overline{0}) \xrightarrow{6,\mu_2} (\overline{0},\overline{(n-1)},\overline{0})$

There exists the following correspondence in my system $((0,0),n,0)$ with actions: τ, $fill$, τ, which have the same update vectors.
And these are the three possible state transitions:

1. $((0,0),n,0) \xrightarrow{\tau,\lambda_1} ((1,0),n,0)$

2. $((0,0),n,0) \xrightarrow{fill,\lambda_3} ((0,1),n,1)$

3. $((0,0),n,0) \xrightarrow{\tau,\mu_2} ((0,0),n-1,0)$

Case 4
From state:
$(\overline{0},\overline{0},\overline{n})$

We can have only three possible actions with these values: Action 1, Action 2 and Action 7.
And these are the three possible state transitions:

1. $(\overline{0},\overline{0},\overline{n}) \xrightarrow{1,\lambda_1} (\overline{1},\overline{0},\overline{n})$

2. $(\overline{0},\overline{0},\overline{n}) \xrightarrow{2,\lambda_3} (\overline{0},\overline{0},\overline{(n+1)})$

3. $(\overline{0}, \overline{0}, \overline{n}) \xrightarrow{7,\mu_3} (\overline{0}, \overline{0}, \overline{(n-1)})$

There exists the following correspondence in my system $((0,n),0,n)$ with actions: τ, $fill$, $empty$ which have the same update vectors.
And these are the three possible state transitions:

1. $((0,n),0,n) \xrightarrow{\tau,\lambda_1} ((1,n),0,n)$

2. $((0,n),0,n) \xrightarrow{fill,\lambda_3} ((0,n+1),0,n+1)$

3. $((0,n),0,n) \xrightarrow{empty,\mu_3} ((0,n-1),0,n-1)$

Case 5
From state:
$(\overline{n_1}, \overline{n_2}, \overline{0})$

We can have only five possible actions with these values: Action 1, Action 2, Action 3, Action 4 and Action 6.
And these are the five possible state transitions:

1. $(\overline{n_1}, \overline{n_2}, \overline{0}) \xrightarrow{1,\lambda_1} (\overline{n_1+1}, \overline{n_2}, \overline{0})$

2. $(\overline{n_1}, \overline{n_2}, \overline{0}) \xrightarrow{2,\lambda_3} (\overline{n_1}, \overline{n_2}, \overline{1})$

3. $(\overline{n_1}, \overline{n_2}, \overline{0}) \xrightarrow{3,\mu_1 p} (\overline{n-1}, \overline{n_2+1}, \overline{0})$

4. $(\overline{n_1}, \overline{n_2}, \overline{0}) \xrightarrow{4,\mu_1(1-p)} (\overline{n_1-1}, \overline{n_2}, \overline{0})$

5. $(\overline{n_1}, \overline{n_2}, \overline{0}) \xrightarrow{6,\mu_2} (\overline{n_1}, \overline{n_2-1}, \overline{0})$

There exists the following correspondence in my system $((n_1,0),n_2,0)$ with actions: τ, $fill$, a, a_0, τ which have the same update vectors.
And these are the five possible state transitions:

1. $((n_1,0),n_2,0) \xrightarrow{\tau,\lambda_1} ((n_1+1,0),n_2,0)$

2. $((n_1,0),n_2,0) \xrightarrow{fill,\lambda_3} ((n_1,1),n_2,1)$

3. $((n_1,0),n_2,0) \xrightarrow{a,\mu_1 p} ((n_1-1,0),n_2+1,0)$

4. $((n_1,0),n_2,0) \xrightarrow{a_0,\mu_1(1-p)} ((n_1-1,0),n_2,0)$

5. $((n_1, 0), n_2, 0) \xrightarrow{\tau,\mu_2} ((n_1, 0), n_2 - 1, 0)$

Case 6
From state:
$(\overline{n_1}, \overline{0}, \overline{n_3})$

We can have only five possible actions with these values: Action 1, Action 2, Action 3, Action 5 and Action 7.
And these are the five possible state transitions:

1. $(\overline{n_1}, \overline{0}, \overline{n_3}) \xrightarrow{1,\lambda_1} (\overline{n_1 + 1}, \overline{0}, \overline{n_3})$

2. $(\overline{n_1}, \overline{0}, \overline{n_3}) \xrightarrow{2,\lambda_3} (\overline{n_1}, \overline{0}, \overline{n_3 + 1})$

3. $(\overline{n_1}, \overline{0}, \overline{n_3}) \xrightarrow{3,\mu_1 p} (\overline{n - 1}, \overline{1}, \overline{n_3})$

4. $(\overline{n_1}, \overline{0}, \overline{n_3}) \xrightarrow{5,\mu_1(1-p)} (\overline{n_1 - 1}, \overline{1}, \overline{n_3 - 1})$

5. $(\overline{n_1}, \overline{0}, \overline{n_3}) \xrightarrow{7,\mu_3} (\overline{n_1}, \overline{0}, \overline{n_3 - 1})$

There exists the following correspondence in my system $((n_1, n_3), 0, n_3)$ with actions: τ, $fill$, a, a_1, $empty$ which have the same update vectors.
And these are the five possible state transitions:

1. $((n_1, n_3), 0, n_3) \xrightarrow{\tau,\lambda_1} ((n_1 + 1, n_3), 0, n_3)$

2. $((n_1, n_3), 0, n_3) \xrightarrow{fill,\lambda_3} ((n_1, n_3 + 1), 0, n_3 + 1)$

3. $((n_1, n_3), 0, n_3) \xrightarrow{a,\mu_1 p} ((n_1 - 1, n_3), 1, n_3)$

4. $((n_1, n_3), 0, n_3) \xrightarrow{a_1,\mu_1(1-p)} ((n_1 - 1, n_3 - 1), 1, n_3 - 1)$

5. $((n_1, n_3), 0, n_3) \xrightarrow{empty,\mu_3} ((n_1, n_3 - 1), 0, n_3 - 1)$

Case 7
From state:
$(\overline{0}, \overline{n_2}, \overline{n_3})$

We can have only four possible actions with these values: Action 1, Action 2, Action 6 and Action 7.
And these are the four possible state transitions:

1. $(\overline{0}, \overline{n_2}, \overline{n_3}) \xrightarrow{1,\lambda_1} (\overline{1}, \overline{n_2}, \overline{n_3})$

2. $(\overline{0}, \overline{n_2}, \overline{n_3}) \xrightarrow{2,\lambda_3} (\overline{0}, \overline{n_2}, \overline{n_3+1})$

3. $(\overline{0}, \overline{n_2}, \overline{n_3}) \xrightarrow{6,\mu_2} (\overline{0}, \overline{n_2-1}, \overline{n_3})$

4. $(\overline{0}, \overline{n_2}, \overline{n_3}) \xrightarrow{7,\mu_3} (\overline{0}, \overline{n_2}, \overline{n_3-1})$

There exists the following correspondence in my system $((0, n_3), n_2, n_3)$ with actions: τ, $fill$, a, τ, $empty$ which have the same update vectors.
And these are the four possible state transitions of state:

1. $((0, n_3), n_2, n_3) \xrightarrow{\tau,\lambda_1} ((1, n_3), n_2, n_3)$

2. $((0, n_3), n_2, n_3) \xrightarrow{fill,\lambda_3} ((0, n_3+1), n_2, n_3+1)$

3. $((0, n_3), n_2, n_3) \xrightarrow{\tau,\mu_2} ((0, n_3), n_2-1, n_3)$

4. $((0, n_3), n_2, n_3) \xrightarrow{empty,\mu_3} ((0, n_3-1), n_2, n_3-1)$

Case 8
From state:
$(\overline{n_1}, \overline{n_2}, \overline{n_3})$

We can have only six possible actions with these values: Action 1, Action 2, Action 3, Action 5, Action 6 and Action 7.
And these are the six possible state transitions:

1. $(\overline{n_1}, \overline{n_2}, \overline{n_3}) \xrightarrow{1,\lambda_1} (\overline{n_1+1}, \overline{n_2}, \overline{n_3})$

2. $(\overline{n_1}, \overline{n_2}, \overline{n_3}) \xrightarrow{2,\lambda_3} (\overline{n_1}, \overline{n_2}, \overline{n_3+1})$

3. $(\overline{n_1}, \overline{n_2}, \overline{n_3}) \xrightarrow{3,\mu_1 p} (\overline{n-1}, \overline{n_2+1}, \overline{n_3})$

4. $(\overline{n_1}, \overline{n_2}, \overline{n_3}) \xrightarrow{5,\mu_1(1-p)} (\overline{n_1-1}, \overline{n_2+1}, \overline{n_3-1})$

5. $(\overline{n_1}, \overline{n_2}, \overline{n_3}) \xrightarrow{6,\mu_2} (\overline{0}, \overline{n_2-1}, \overline{n_3})$

6. $(\overline{n_1}, \overline{n_2}, \overline{n_3}) \xrightarrow{7,\mu_3} (\overline{0}, \overline{n_2}, \overline{n_3-1})$

There exists the following correspondence in my system $((n_1, n_3), n_2, n_3)$ with actions: τ, $fill$, a, a_1, τ, $empty$ which have the same update vectors.
.
And these are the six possible state transitions:

1. $((n_1, n_3), n_2, n_3) \xrightarrow{\tau, \lambda_1} ((n_1 + 1, n_3), n_2, n_3)$

2. $((n_1, n_3), n_2, n_3) \xrightarrow{fill, \lambda_3} ((n_1, n_3 + 1), n_2, n_3 + 1)$

3. $((n_1, n_3), n_2, n_3) \xrightarrow{a, \mu_1 p} ((n_1 - 1, n_3), n_2 + 1, n_3)$

4. $((n_1, n_3), n_2, n_3) \xrightarrow{a_1, \mu_1 (1-p)} ((n_1 - 1, n_3 - 1), n_2 + 1, n_3 - 1)$

5. $((n_1, n_3), n_2, n_3) \xrightarrow{\tau, \mu_2} ((n_1, n_3), n_2 - 1, n_3)$

6. $((n_1, n_3), n_2, n_3) \xrightarrow{empty, \mu_3} ((n_1, n_3 - 1), n_2, n_3 - 1)$

We have shown that $\forall$ states $(\overline{n_1}, \overline{n_2}, \overline{n_3})$ there exists a correspondence to a state $((n_1, n_3), n_2, n_3)$ and equivalent updates are applied to them. Moreover, they can perform the same actions.
Since the actions done by my system are the all and only ones permitted in those cases the system represents all and only the states of original G-Network with trigger, and also their behaviours are equivalent.

We can now state that with the proposed encoding of this system, **we have modelled a G-Network using only the definitions of the PEPA language**. Thus, we introduced a method to extend its expressivity to cover also this kind of G-Network.

4.3.5 Product-Form of G-Network with Trigger

We will now analyse the derivation graph of Double Index solution and search where it doesn't satisfy the conditions of the product-form theorem. After that, we modify it in order to satisfy those conditions but without changing its behaviour that we already proved it is equal to the original one.
We add a new state in the processes, the *Phantom State* F. The purpose of this stratagem is in fact to satisfy the two conditions without changing the behaviour thanks to its property of unreachability. The main property of state F is exactly that *from the starting normal states, every possible and permissible transaction in the system cannot lead in any way to the phantom state* F.

4.3.5.1 Conditions for the Product-Form Theorem

In order to have the product-form solution, a system in PEPA must satisfy the following conditions:

1. Each passive action must be outgoing from each state of processes in which it is used at least one time, i.e., each state must have an outgoing transition of type of the passive action and with $\top$ rate.

2. Each active action must be ingoing to each state of processes in which it is used at least one time, i.e., each state must have an ingoing transition of type of the active action and with a non-$\top$ rate.

3. All reverse rate of the same action type in a process must be equal, i.e., all the reverse rates of a passive action with $\top$ rate within a process must be all equal.

Another important fact is that all these referred actions are in a cooperation set.
To satisfy these conditions we will adopt the following techniques:

1. Outgoing Passive Actions: we add an exiting transition from all states that don't have it, of that type of passive action (it doesn't matter if it is a self-loop).

2. Ingoing Active Actions: we add an entering transition to all states that don't have it, of that type of active action (it doesn't matter if it is a self-loop).

3. We will balance the equations of my processes and after that we compute the reverse rates of all the passive actions.

Analysing this graph in Figure 4.3 we have the following information regarding action types:

- Active Actions: a, a_0, a_1 (τ doesn't cooperate with anyone)

- Passive Actions: $fill$, $empty$

And the following information is about the first two conditions of the product-form theorem:

- State P_0:
  Ingoing Active Actions: a, a_0, a_1
  Missing Active Actions: none
  Outgoing Passive Actions: $fill$
  Missing Passive Actions: $empty$

- State P_n:
  Ingoing Active Actions: a, a_0, a_1
  Missing Active Actions: none
  Outgoing Passive Actions: $fill$
  Missing Passive Actions: $empty$

- State P_0^i:
  Ingoing Active Actions: a, a_1
  Missing Active Actions: a_0
  Outgoing Passive Actions: $fill$, $empty$
  Missing Passive Actions: none

- State P_n^i:
  Ingoing Active Actions: a, a_1
  Missing Active Actions: a_0
  Outgoing Passive Actions: $fill$, $empty$
  Missing Passive Actions: none

Analysing the graph in Figure 4.4 we have the following information regarding action types:

- Active Actions: none (τ doesn't cooperate with anyone)
- Passive Actions: a, a_1

And the following information is about the first two conditions of the product-form theorem:

- State Q_0:
  Ingoing Active Actions: none
  Missing Active Actions: none
  Outgoing Passive Actions: a, a_1
  Missing Passive Actions: none

- State Q_n:
  Ingoing Active Actions: none
  Missing Active Actions: none
  Outgoing Passive Actions: a, a_1
  Missing Passive Actions: none

Analysing the graph in Figure 4.5 we have the following information regarding action types:

- Active Actions: $fill$, $empty$
- Passive Actions: a_0, a_1

And the following information is about the first two conditions of the product-form theorem:

- State R_0:
  Ingoing Active Actions: $empty$
  Missing Active Actions: $fill$
  Outgoing Passive Actions: a_0
  Missing Passive Actions: a_1

- State R_n:
  Ingoing Active Actions: $fill$, $empty$
  Missing Active Actions: none

Outgoing Passive Actions: a_1
Missing Passive Actions: a_0

There are some states in which an action has problems in satisfying the conditions.
$(\leftarrow)$ represents an outgoing passive problem, $(\rightarrow)$ represents an ingoing active problem.
This is the summary of the DI solution:

Process	Action a	Action a_0	Action a_1	Action $fill$	Action $empty$
P		$(\rightarrow) : P_0^i, P_n^i$			$(\leftarrow) : P_0, P_n$
Q					
V		$(\leftarrow) : V_n$	$(\leftarrow) : V_0$	$(\rightarrow) : V_0$	

4.3.5.2 Addition of Phantom State and Missing "Conditions" Actions

In this section we add those missing actions to my double index solution. We call them *impossible actions* because they will never occur due to the fact that they will never cooperate with the corresponding active/passive actions in another process. We use the annotation of putting a ! before their name to distinguish them from the normal actions.
In order to do this, we add a state F, called *Phantom state*. This state will never be reachable, because all the actions which lead to it are impossible actions. In this way all the actions from the unreachable state F will be impossible too, because they will never occur thanks to the non-reachability of F.
The possible solutions are depicted in Figure 4.6 and 4.7:

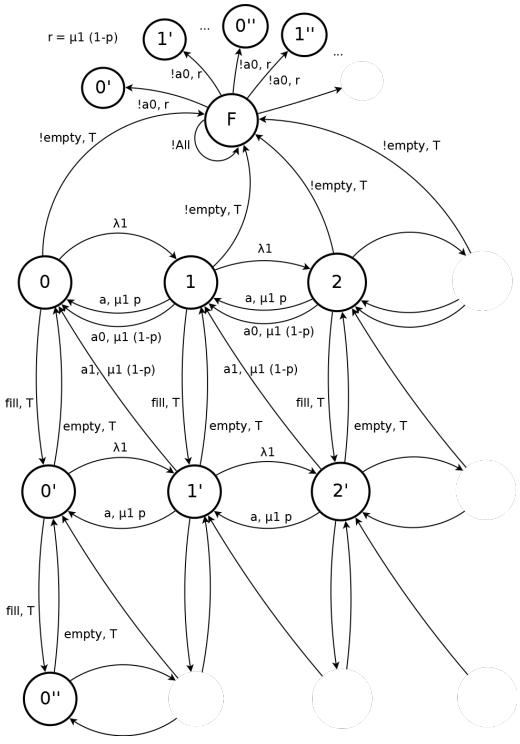


Figure 4.6: R_1 with impossible actions (DI)

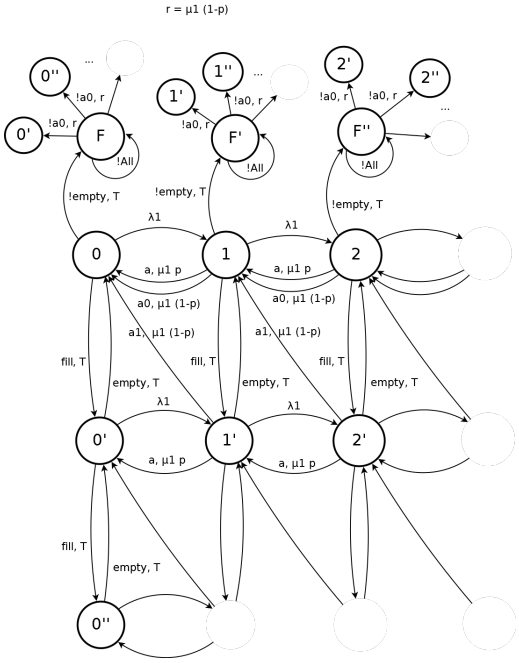


Figure 4.7: R_1 with impossible actions (DI), alternative

We can see that F is unreachable because in state P_0, the process R_0 doesn't have any positive customer and so it cannot perform an $empty$ action. This ensures that $empty$ in P_n with $n \geq 0$ doesn't occur and so F is unreachable.
We add also in the phantom state, the action All which represent all possible actions, in this way the first two conditions of product-form theorem are satisfied also for F state. However, they are still impossible.
A solution could be even splitting the F state in more phantom states. The meaning is the same, F states are still unreachable and their ingoing and outgoing actions still impossible. In this way we show that it is the same to have a single phantom state F or a set of them $(F_0, F_1, ..., F_n)$. Moreover, each F state has its action All and like in the previous example, they satisfy the two conditions even if they are still impossible. Another important fact is that *we can add whichever action from any phantom state F_i to any other phantom state F_j* (included $i = j$ with a self-loop) *because they will be still impossible*.
In both solutions we add the passive $empty$ action in all P_n with $n \geq 0$ and a_0 in all P_n^i with $n \geq 0$ and $i > 0$.
Similarly we can do the same for R_3 in Figure 4.8 and 4.9
A possible alternative in this case is the use of only one F phantom state for the missing actions of state 0 and transform all the others outing missing actions of states $1, ..., n$ into self-loops. The state 0 needs to have an ingoing missing action and in order to not synchronise with anyone we must add a phantom state F in this case in contrast to other states.

The definition of an **impossible action** is:
An action whose type is part of a cooperation set and never synchronised with anyone (i.e., it will never occur) is called an impossible action.
This means that impossible actions will never cooperate with their corresponding active or passive actions in other processes and they are usually marked with a ! before their

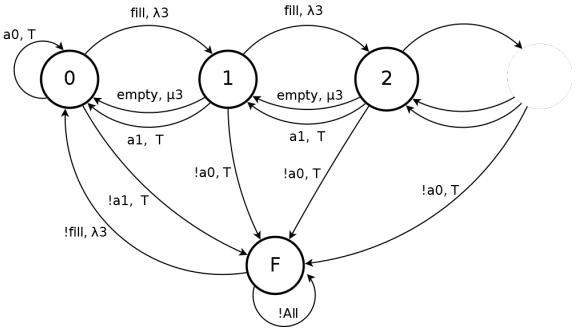


Figure 4.8: R_3 with impossible actions (DI)

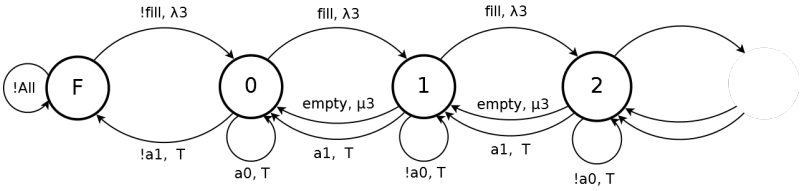


Figure 4.9: R_3 with impossible actions (DI), alternative

name to discern them from the normal actions.

To better understand them, we make an example. Considering the following encoding of a system in PEPA:

$$P = \begin{cases} P_0 = (a, \top).P_1 \\ P_1 = (b, \mu_1).P_0 + (c, \mu_2).P_0 \end{cases}$$

$$Q = \begin{cases} Q_0 = (a, \mu).Q_1 + (c, \top).Q_1 \\ Q_1 = (b, \top).Q_0 \end{cases}$$

$$S \stackrel{def}{=} P \underset{a,b,c}{\bowtie} Q$$

And their derivations graphs are the following two:
We can notice that all actions are in the cooperative set and a,b have the same direction in both processes (from 0 to 1 and from 1 to 0, respectively) but c has an opposite behaviour ($1 - 0$ direction in P and $0 - 1$ direction in Q). Furthermore a is passive in P and active in Q and b,c are active in P and passive in Q.
In this way, considering that the starting states are P_0 and Q_0 (or we will incur in a deadlock situation), when the two processes cooperate in general system S we have the following states:

- General State S_0: corresponds to the states of processes P_0 and Q_0
- General State S_1: corresponds to the states of processes P_1 and Q_1

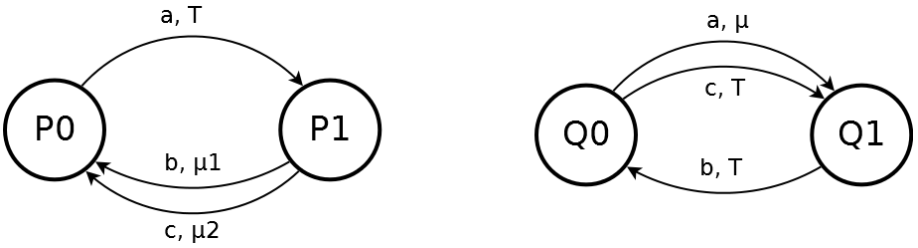


Figure 4.10: Example of Impossible Action c in processes P and Q

Moreover, starting from the cooperation of P_0 with Q_0, process S will have the following behaviour:

- In S_0:
  - P_0 has only a passive action a and must wait for Q_0
  - Q_0 could perform a and c but both must synchronise with the same type of actions in P_0
  - Result: Q_0 can only synchronise in action a with P_0 and it proceeds in Q_1 and similarly P_0 proceeds in P_1 always performing action a

  In this way S_0 becomes state S_1 and c in this case can't occur.

- In S_1:
  - P_1 could perform b and c but both must synchronise with the same type of actions in Q_1
  - Q_1 has only a passive action b and must wait for P_1
  - Result: P_1 can only synchronise in action b with Q_1 and it proceeds in P_0 and similarly Q_1 proceeds in Q_0 always performing action b

  In this way S_1 becomes state S_0 and c also in this case can't occur.

- We return to state S_0 so no other behaviours can occur.

With this behaviour, the derivative graph of S is the following:

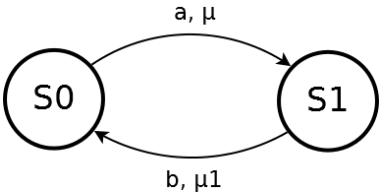


Figure 4.11: Example of Impossible Action c in cooperation process of P and Q

We can see how c never occurs in both directions (i.e., either from S_0 to S_1 or from S_1 to S_0). In this case c never synchronises and subsequently never occurs so in this system both c actions are *impossible actions*.

The definition of a **phantom state** F is:
A non-starting state in which all ingoing actions are impossible actions is called a phantom state.
This means that the entering actions in a phantom state F can be of three types (even if still impossible):

- Impossible actions from other kind of states
- Self-loops of impossible actions
- Impossible actions from other phantom states

This definition ensures that a phantom state is unreachable and not present in the derivative set of the process because all the actions which lead to it are impossible actions. In this way all outgoing actions of F are impossible too because of the state will never be a future state of the process and subsequently all its actions will never occur.
So, due to the unreachability of phantom state F, *all the actions, both ingoing and outgoing, of a phantom state F are consequently impossible actions*.
This leads the definition of a **set of phantom states** F:
A set of states in which there isn't any starting state and all actions (ingoing and outgoing) from outside the set are impossible actions is called a set of phantom states $F = (F_1, ..., F_n)$.
Similarly to the previous definition all outgoing actions from a set of phantom states are still impossible actions and they can be of three type:

- Actions to outside the set
- Self-loops of actions
- Impossible actions between phantom states

To better understand them, we make an example. Considering the following encoding of a system in PEPA:

$$P = \begin{cases} P_0 = (a, \top).P_1 + (b, \mu_1).F_P \\ P_1 = (b, \mu_1).P_0 + (c, \mu_2).F_P \\ F_P = (a, \top).P_1 \end{cases}$$

$$Q = \begin{cases} Q_0 = (a, \mu).Q_1 + (c, \top).F_Q \\ Q_1 = (b, \top).Q_0 + (a, \mu).F_Q \\ F_Q = (a, \mu).Q_1 \end{cases}$$

$$S \stackrel{def}{=} P \underset{a,b,c}{\bowtie} Q$$

And their derivations graphs are the following two:
We can notice that all actions are in the cooperation set but apart from a from P_0 and Q_0 and b from P_1 and Q_1, the others actions don't have the same direction in both processes. Furthermore a is passive in P and active in Q and b,c are active in P and passive in Q.
In this way, considering that the starting states are P_0 and Q_0 (or we will incur a dead-lock situation), when the two processes cooperate in general system S we have the following states:

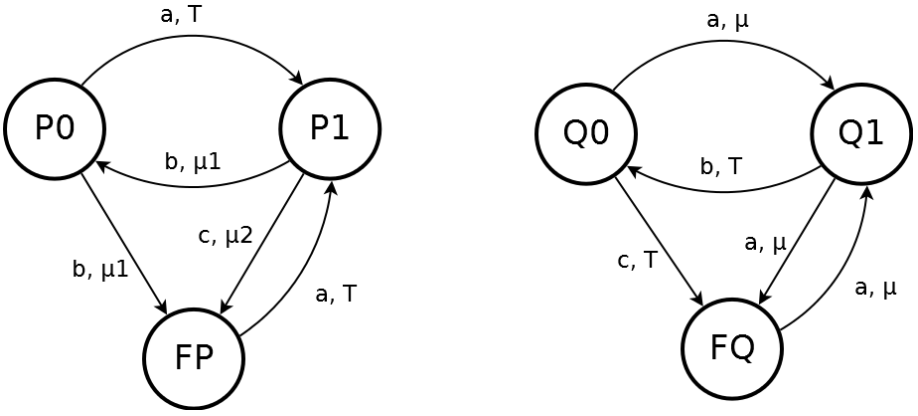


Figure 4.12: Example of Phantom State F in processes P and Q

- General State S_0: corresponds to the states of processes P_0 and Q_0
- General State S_1: corresponds to the states of processes P_1 and Q_1

Moreover, starting from the cooperation of P_0 with Q_0, process S will have the following behaviour:

- In S_0:
  - P_0 could perform a and b but both must synchronise with the same type of actions in Q_0
  - Q_0 could perform a and c but both must synchronise with the same type of actions in P_0
  - Result: Q_0 can only synchronise in action a with P_0 and it proceeds in Q_1 and similarly P_0 proceeds in P_1 always performing action a

  In this way S_0 becomes state S_1 and b and c in this case can't occur.
- In S_1:
  - P_1 could perform b and c but both must synchronise with the same type of actions in Q_1
  - Q_1 could perform b and a but both must synchronise with the same type of actions in P_1
  - Result: P_1 can only synchronise in action b with Q_1 and it proceeds in P_0 and similarly Q_1 proceeds in Q_0 always performing action b

  In this way S_1 becomes state S_0 and a and c also in this case can't occur.
- We return to state S_0 so no other behaviours can occur.

With this behaviour, the derivative graph of S is the following:
The final result process is the same in the two previous examples, this means that both impossible actions and phantom state F didn't affect the main system S.
It is important to notice that F_P and F_Q could have synchronised and performed action a but since P, Q and in general S have never reached both states F_P and F_Q then their a actions never synchronise and occur so they are impossible actions.
We can see how F is never reached from any state (i.e., neither from S_0 nor from S_1). In this case no action reaches F so in this systems both F_P and F_Q states are *phantom states*.

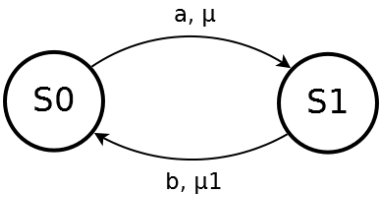


Figure 4.13: Example of Phantom State F in cooperation process of P and Q

4.4 Conclusions

We have modelled a case study with G-Network with signals in which customers can be moved from one queue to another upon the arrival of a signal. With our work, we introduced an encoding method, called Double Index (DI) method, for modelling G-Networks using the PEPA language. In order to satisfy the conditions of the product-form theorem we have to add to the system in PEPA also the notions of *Phantom State* and *Impossible Actions*. In this way even if we change the structure of the system we have maintained its behaviour because the new states will never be reachable since all the actions which lead to them are impossible actions. Thus, they help to satisfy the theorem conditions without altering the behaviour of the overall system.

Chapter 5

Analysis of Systems with Ageing Objects

5.1 Introduction

In this chapter we will propose a new model for the analysis of systems with ageing objects such as Time-To-Live cache. We consider a model with an underlying Continuous Time Markov Chain in which objects can be completely or partially rejuvenated. In the former case just one object becomes fresh, while in the latter all the objects are simultaneously rejuvenated so that the youngest becomes fresh. We show that under the so-called Independent Reference Model assumption our model is numerically tractable and has a product-form equilibrium distribution. Furthermore, we consider the case in which the object ageing stops after a certain threshold and hence the partial rejuvenation introduces a probabilistic behaviour. Also in this case, we can derive a product-form equilibrium distribution under some mild conditions. The models presented in our work may be interpreted as a new class of G-Networks with catastrophes and partial flushing.

5.1.1 TTL caches

Stochastic models are powerful tools for assessing the non-functional quantitative properties of computer networks, communication systems, and software architectures. In many practical applications, Markov processes are the stochastic processes underlying the considered models and their performance evaluation is carried out by using the well-known methods for the analysis of transient or stationary behaviour of Markov processes. We will now focus on the analysis of models whose underlying process is a Continuous Time Markov Chain (CTMC). Since its introduction [22], the theory of product-form solutions has played an important role in the practical analysis of models with underlying CTMCs as it allows for an efficient derivation of the stationary performance indices even when the process's state space is huge and the analysis methods based on the solution of the system of global balance equations become computationally prohibitive. Even more interestingly, for a class of product-form models, including the ones we are studying here, the performance indices can be derived without even generating the joint state space. Successful applications of product-form theory include the BCMP theorem [14], the modelling of neural networks [51], the analysis of systems with fork and join constructs [84], the loss networks [76] and the performance evaluation of wireless networks [17], just to mention few.

We now focus on modelling and analysing systems with ageing objects by means of product-form models. These systems consist of a set of objects which are associated with an age (e.g., the time-stamp of their creation or latest access). As time passes, the objects become older. Two types of events can rejuvenate the objects:

- total rejuvenation, i.e., the object timestamp is set to the current time. This event affects only one object.
- partial rejuvenation. In this case the event affects the whole system since the objects are all rejuvenated for the same time interval so that the youngest is associated with the current timestamp.

An example of such a system is a TTL cache in which the total rejuvenation occurs when an object is accessed and the partial rejuvenation can be seen as a method to prevent an under utilisation of the cache memory in case of periods of inactivity. The networking research community has renewed its interest in the performance of caching systems due to the new delivery methods for distributing contents in the networks. The huge number of proxy servers has led to the design of Content Delivery Networks (CDN) which are used by the content providers to deliver information in a large and spread population of users. Caching contents that have the greatest demand closer to the users' locations allow one to improve the client-perceived experience, to reduce the server load and optimize the bandwidth requirements. In this perspective, the caching system plays a fundamental role in the gradual shift from the traditional paradigm of host-to-host communication to the new host-to-content model. Other applications of ageing systems are shown in [53, 56] where the failure of nodes in distributed systems are handled by means of checkpoints.

Technical contributions and related work

The main contributions with respect to the literature are the following:

- We present two models in product-form for the performance evaluation of systems with ageing objects. The main difference between the two models is that one allows the object age to grow indefinitely, while the other introduces a maximum age threshold. We discuss the implications on practical applications with some examples. The product-form analysis that we demonstrate is interesting for at least two aspects. The first is that neither the joint CTMC nor the CTMC underlying a single model are reversible as it happens, e.g., in Jackson's queues [74] and G-queues [48]. The second interesting aspect is that synchronisations among objects are not pairwise, i.e., at a given epoch more than two objects can simultaneously change their states. There are few results in this direction in the literature of product-forms. In [50, 38] the authors consider queueing networks in which the departure of a customer from a queue causes a movement of one job from a second queue to a third one, hence causing the simultaneous state change of three components. However, the extension of the result to more than three components is not trivial mainly because the proof technique adopted in those papers is based on solving the system of global balance equations (GBEs). The first model we propose is in the style of G-Networks as proposed in [41], while the one with maximum ageing is, to the best of our knowledge, very peculiar since very few product-forms are known for finite state space models [3, 6]. The contribution of the unbounded model with respect to [41] is twofold. First, the proof is not based on the solution of the system of global

balance equations of the joint model. Secondly, we consider individual jumps of the objects to the zero state. Our proof method is based on the quasi-reversibility property [76] and the Reversed Compound Agent Theorem (RCAT) [61, 7]. Both these results provide a way to elegantly prove the product-form of a CTMC but they consider only pairwise synchronisations and hence they cannot be straightforwardly applied to study our models. We show that they can still be used by introducing a passage to the limit for a transition rate in a similar fashion to what has been done in [27, 65, 84]. Proofs of product-forms based on quasi-reversibility are simple to handle and compositional in the sense that they allow the combination of the models that we study here with others which are known to be quasi-reversible while maintaining the product-form of the equilibrium distribution. As a consequence, heterogeneous networks may be studied without constructing the joint Markov chain.

- We show how to numerically derive the models' performance indices without constructing the joint CTMCs. This is important because the structures of these chains can be complex since the transitions corresponding to partial rejuvenations depend on the global state of the models. The derivation of the performance indices requires us to solve a non-linear system of equations. We propose a fixed point algorithm to tackle this problem and show its efficiency and convergence properties on numerous examples. The system of equations admits a unique positive solution. With respect to [41, 42], we do not require any modification of the network of objects in order to obtain the convergence of the algorithm.

- As an example we apply our model for the analysis of TTL cache with partial rejuvenation. First we propose an ideal model, whose implementation is very expensive, in which a timer is associated with each object despite the fact that it is inside or outside the cache. We study the performance indices under the Independent Reference Model (IRM) assumptions [45, 77, 108]. Then, we consider a model in which we maintain the timers only for the objects inside the cache. The partial rejuvenation of objects outside the cache has a probabilistic effect, i.e., the object may remain outside or can be copied inside the cache according to a Bernoulli trial. We prove that it is possible to obtain exactly the same expected performance indices of the ideal model while maintaining the product-form property. We discuss how it is possible to dynamically set the model's parameters to achieve some performance goals. The analyses of TTL caches, often connected to form networks, have been widely addressed in recent years (see, e.g., [15, 34, 35] and the references therein). In our case study we consider a simpler situation of a single cache as in [91]. Clearly, the analysis becomes challenging because of the partial rejuvenation signals which aim to avoid the under utilisation of the cache.

We now give some theoretical background and introduce the notation for our model with unbounded ageing and then prove its product-form.

5.1.2 Preliminaries

Let $X(t)$ be a stationary CTMC on the state space $\mathcal{S}$. Its reversed process, denoted $X^R(t)$, is still a stationary CTMC [61, 76] whose transition rates are defined as follows:

$$q^R(s_1, s_2) = \frac{\pi(s_2)}{\pi(s_1)} q(s_2, s_1)\,, \tag{5.1}$$

where $q(s_2, s_1)$ is the transition rate from state s_2 to s_1 in $X(t)$ and $q^R(s_1, s_2)$ its inverse in $X^R(t)$. $X(t)$ is reversible if it is stochastically indistinguishable from $X^R(t)$. Henceforth we assume that $X(t)$ is ergodic, since the models we are presenting in the following sections are all unconditionally ergodic. The equilibrium distributions π of $X(t)$ and $X^R(t)$ are identical. Moreover, given the forward and the reversed chain, the following generalised Kolmogorov's criteria hold.

Proposition 2 (Kolmogorov's generalised criteria [61])**.** Let $X(t)$ be an ergodic CTMC with state space $\mathcal{S}$ and infinitesimal generator $\mathbf{Q}$, then $Y(t)$ with the same state space and infinitesimal generator $\mathbf{Q}'$ is the reversed process $X^R(t)$ if and only if:

- For every state $s_1 \in \mathcal{S}$ we have:

$$\sum_{\substack{s_2 \in \mathcal{S} \\ s_2 \neq s_1}} q(s_1, s_2) = \sum_{\substack{s_2 \in \mathcal{S} \\ s_2 \neq s_1}} q'(s_1, s_2)\,,$$

  i.e., the residence times in a state in the forward and in the reversed processes have the same distribution.

- For every finite sequence of states $s_1, s_2, \ldots, s_n \in \mathcal{S}$, it holds that:

$$q(s_1, s_2)q(s_2, s_3)\cdots q(s_n, s_1) = q'(s_1, s_n)q'(s_n, s_{n-1})\cdots q'(s_2, s_1)\,.$$

In the analysis of the model presented in Section 5.1.3 we will use Proposition 2 to derive the rates of the reversed chain of the CTMC underlying the model.
We should stress that one can derive the reversed process $X^R(t)$ for any stationary chain $X(t)$ even if this is not reversible. Indeed, we will widely base the product-form analysis presented in Sections 5.1.3 and 5.2.4 on the derivation of reversed processes of the CTMCs underlying the proposed models even if these chains are *not* reversible.
If one knows the infinitesimal generator of both the forward and the reversed chain, $\mathbf{Q}$ and $\mathbf{Q}^R$, respectively, then we can compute the expression of the equilibrium distribution in a very efficient way. Indeed, it suffices to choose an arbitrary reference state $s_0 \in \mathcal{S}$ and then to compute the equilibrium probability of any state $s \in \mathcal{S}$ with respect to s_0 by finding a path from s_0 to s, e.g.,

$$s_0 \to s_1 \to s_2 \to \cdots \to s_n = s\,.$$

Then, we have (see, e.g., [85, 87])

$$\pi(s) = \pi(s_0)\frac{\prod_{i=1}^{n} q^R(s_i, s_{i-1})}{\prod_{i=1}^{n} q(s_{i-1}, s_i)}\,. \tag{5.2}$$

Finally, we recall that quasi-reversibility [27, 76] is a sufficient condition to ensure that the synchronisation of a set of models whose underlying stochastic process is an ergodic CTMC has product-form solution. Informally, we can think that the transition in one of the components can trigger a transition in another by sending a signal. In the terminology used in [76] we say that there is a departure from the component sending the signal and an arrival at the one receiving the signal, while in the process algebraic terminology [70], we say that the component which sends the signal is active and the one receiving it is passive. In the analysis of quasi-reversible models, the receiver cannot prevent the transitions in the sender (but it may ignore them). For the sake of simplicity, we assume that a component sends a synchronising signal only to another component (that will be the case for the models we study in the following sections).

Proposition 3 (Quasi-reversibility)**.** Given a cooperating component let $\mathcal{T} \subset \mathcal{S} \times \mathcal{S}$ be the set of transitions that synchronise with another model. Then, the model is quasi-reversible if there exists $x \in \mathbb{R}^+$ such that for all the states $s \in \mathcal{S}$, we have:

$$x = \frac{\sum_{s':(s',s)\in\mathcal{T}} \pi(s')q(s',s)}{\pi(s)}\,.$$

It is well-known that quasi-reversibility is a sufficient condition for the existence of a product-form equilibrium distribution of the joint model [61, 76, 90].

5.1.3 A model for ageing objects

5.1.3.1 The stochastic process underlying the collection of ageing objects.

We consider a set of K objects whose age at time t is modelled by a stochastic process $Y_k(t)$, $1 \le k \le K$, which takes values in $\mathbb{N}$. Let $\mathbf{Y}(t) = (Y_1(t), \ldots, Y_K(t))$ be the stochastic process associated with all the objects and let $\mathbf{u} = (u_1, \ldots, u_K)$ be its state at time t, and $u_{\min} = \min(u_i, 1 \le i \le K)$. We characterise the stochastic process $\mathbf{Y}(t)$, for $h \to 0^+$, as:

$$\begin{aligned}
&Pr\{\mathbf{Y}(t+h) = \mathbf{u} - \mathbf{1}u_{\min} | \mathbf{Y}(t) = \mathbf{u}\} = \eta h + o(h)\\
&Pr\{\mathbf{Y}(t+h) = \mathbf{u}[u_k \leftarrow 0] | \mathbf{Y}(t) = \mathbf{u}\} = \lambda_k h + o(h)\\
&Pr\{\mathbf{Y}(t+h) = \mathbf{u}[u_k \leftarrow u_k + 1] | \mathbf{Y}(t) = \mathbf{u}\} = \gamma_k h + o(h)\\
&Pr\{\mathbf{Y}(t+h) = \mathbf{u} | \mathbf{Y}(t) = \mathbf{u}\} = 1 - \left(\eta + \sum_{k=1}^{K} \lambda_k + \gamma_k\right) h + o(h)
\end{aligned}$$

where $\mathbf{u}[u_k \leftarrow val]$ denotes vector $\mathbf{u}$ in which component k takes value val, and $\mathbf{1}$ is a vector of 1s with size K. We say that λ_k is the *refresh rate* for object k, η is the partial rejuvenation rate and γ_k is the ageing rate. Clearly, $\mathbf{Y}(t)$ is a CTMC.
We are interested in computing the equilibrium distribution of $\mathbf{Y}(t)$. Notice that the partial rejuvenation events are such that processes $Y_k(t)$ are not stochastically independent. Moreover, $\mathbf{Y}(t)$ is neither reversible nor in product-form, and hence the computation of the equilibrium distribution can be prohibitive for a large number of objects even if truncation is applied.

5.1.4 A product-form approximation for $\mathbf{Y}(t)$

In this section we approximate the CTMC $\mathbf{Y}(t)$ introduced in Section 5.1.3.1 by a CTMC $\mathbf{X}(t)$. We introduce a different semantics for the partial rejuvenation signals that allows us to prove that $\mathbf{X}(t)$ has a product-form equilibrium distribution. Let us introduce the following notation:

$$k^+ = \begin{cases} k+1 & \text{if } k < K \\ 1 & \text{if } k = K \end{cases} \qquad k^- = \begin{cases} k-1 & \text{if } k > 1 \\ K & \text{if } k = 1 \end{cases}.$$

In $\mathbf{X}(t)$, a partial rejuvenation signal iteratively decreases the age of the objects according to their orders until we find a fresh object, i.e., whose age is 0. Formally, the destination state reached by $\mathbf{X}(t)$ immediately after a partial rejuvenation signal is given by Algorithm 1. The ageing rate and the refresh rate have the same effects as those described for $\mathbf{Y}(t)$.
Notice that the model does not exactly implement the behaviour described for $\mathbf{Y}(t)$ as shown by the following counterexample. Consider a model with $K = 4$ objects, and

suppose their ages are $(2, 1, 3, 4)$ when the reset signal arrives at the first timer. Then, at t_0^+ the state of $\mathbf{Y}(t)$ is $(1, 0, 2, 3)$, however Algorithm 1 performs a first iteration on all the objects and a second one that stops when it reaches the second object leading to a state in t_0^+ which is $(0, 0, 2, 3)$. Notice that the difference between the correct age of an object and the one which is computed by Algorithm 1 can be at most of one unit, therefore we consider the approximation acceptable.

Algorithm 1: Reset of timers upon arrival at time t_0 of a reset signal to the o_κ timer.

Data: κ, $X_i(t_0)$
Result: $X_i(t_0^+)$
for $k \in [1, K]$ **do**
  $X_k(t_0^+) = X_k(t_0)$;
end
$k \leftarrow \kappa$;
while $X_k(t_0^+) > 0$ **do**
  $X_k(t_0^+) \leftarrow X_k(t_0^+) - 1$;
  $k \leftarrow k^+$;
end

5.1.5 Stationary analysis of $\mathbf{X}(t)$

In this section we derive the stationary distribution of the model and prove that it is in product-form and unconditionally stable.

Theorem 5.1. *The stochastic process $\mathbf{X}(t)$ has the product-form equilibrium distribution:*

$$\pi^X(\mathbf{u}) = \prod_{k=1}^{K} \pi_k^X(u_k) = \prod_{k=1}^{K} (1 - \rho_k)\rho_k^{u_k}\,, \tag{5.3}$$

where ρ_k is the solution of the following non-linear system of rate equations:

$$\begin{aligned} \rho_k &= \frac{1}{2(x_k + \eta/K)} \cdot \\ &\cdot \left(\lambda_k + \frac{\eta}{K} + x_k + \gamma_k - \sqrt{\left(\lambda_k + \frac{\eta}{K} + x_k + \gamma_k\right)^2 - 4\gamma_k \left(x_k + \frac{\eta}{K}\right)} \right) \end{aligned} \tag{5.4}$$

$$x_{k^+} = \rho_k \left(x_k + \frac{\eta}{K}\right) \tag{5.5}$$

for all $k = 1, \ldots, K$, and u_k is the age of object k. Moreover, the model is unconditionally stable for strictly positive rates λ_k and γ_k.

In principle, one could prove Theorem 5.1 by substituting Expression (5.3) in the system of the global balance equations for $\mathbf{X}(t)$ in a similar fashion to what has been done in [41, 42]. However, the complexity in the structure of $\mathbf{X}(t)$, which depends on Algorithm 1, makes this way of proving the equilibrium distribution long and prone to errors.

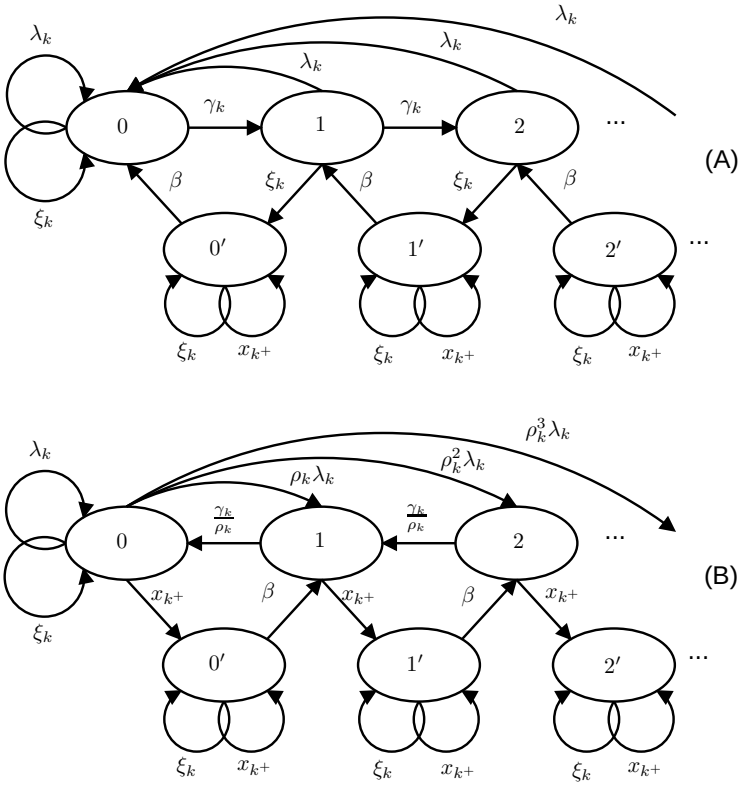


Figure 5.1: Quasi reversible model for the single object ageing. (A)-Forward process and (B)-Reversed process.

Proof. Let us consider the model for the single object ageing depicted in Figure 5.1-(A). In this model the arrivals of partial rejuvenation signals occur with rate ξ_k. At the arrival of such a signal at object k, this moves from state $u > 0$ to state u' and then propagates the signal to object k^+ while moving to state $u - 1$. The signal propagation becomes instantaneous as required by the definition of $\mathbf{X}(t)$ when $\beta \to \infty$. We prove the product-form by resorting to the quasi-reversibility property, i.e., we prove that the occurrences of the transitions from state i' to state $(i-1)'$, $i > 0$, at t_0 are independent of the occurrences of the same transitions subsequent t_0. These are the transitions that forward the partial rejuvenation signal from object k to k^+.

Claim 1. The reversed CTMC of the process shown in Figure 5.1-(A) is the process shown in Figure 5.1-(B) where

$$\rho_k = \frac{1}{2\xi_k}(\lambda_k + \gamma_k + \xi_k - ((\lambda + \gamma_k + \xi_k)^2 - 4\xi_k\gamma_k)^{1/2}$$

and $x_{k^+} = \rho_k \xi_k$.

The claim can be readily verified by applying the generalised Kolmogorov's criteria given in Proposition 2. First, we check that the total rate out of every state in the forward and the reversed process are the same. For states u'_k it is trivial. For state $u_k > 0$ we have:

$$\gamma_k + \lambda_k + \xi_k = \frac{\gamma_k}{\rho_k} + x_{k^+}\,, \tag{5.6}$$

which is satisfied for the definition of ρ_k given in Claim 1. For state 0 we have to prove that $\gamma_k = \sum_{u_k=1}^{\infty} \rho_k^{u_k}\lambda_k + x_{k^+} = \lambda_k\rho_k/(1-\rho_k) + x_{k^+}$ which is equivalent to Equation (5.6).

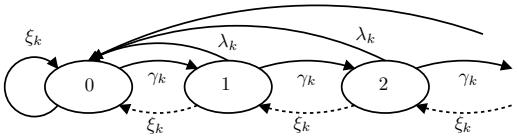


Figure 5.2: Simplified version of the model of Figure 5.1 for proving the quasi-reversibility property.

Finally, we can easily see that the product of the rates in the forward and reversed processes are the same for every minimal cycle. This is sufficient to prove Claim 1. The model is quasi-reversible (and satisfies RCAT conditions) because every state in the reversed process has an outgoing rate of x_{k^+} associated with the propagation of the partial rejuvenation signal. By the definition of the rates in the reversed process, we can derive the equilibrium distribution of state $u_k > 0$ by using the path from u_k to 0 according to Equation (5.2). We have $\pi_k^X(u_k) = \pi_k^X(0)\rho_k^{u_k}$ and we easily see that $\pi_k^X(u'_k) = \pi_k^X(u_k)\xi_k/\beta$. Notice that neither x_{k^+} nor $\pi_k^X(u_k)$ depends on β. Indeed, for $\beta \to \infty$ we have the instantaneous propagation of the partial rejuvenation signal as required and the model is quasi-reversible and $\pi_k^X(u'_k) \to 0$.
Hence, the theorem is proved by noticing that ξ_k is the sum of the rates of the partial rejuvenation signals arriving from k^- and from outside, i.e., $\xi_k = x_k + \eta/K$. □

The proof method based on the passage to the limit for modelling instantaneous propagation of transitions is inspired by the approach used in [27, 65, 84] for different networks and is an alternative to the process algebraic one recently proposed in [66]. Notice that, thanks to the passage to the limit $\beta \to \infty$, proving the product-form of a component such as the one shown in Figure 5.1 can be readily done by considering the simplified model shown in Figure 5.2, in the sense that if the latter is quasi-reversible also the former is quasi-reversible. In the model depicted in Figure 5.2, one has just to prove that the reversed rates of the dotted transitions (those propagating the rejuvenation) are constant [61].

5.1.6 Solving the system of rate equations

The system of rate equations (5.4)-(5.5) does not generally have a symbolic solution and its degree grows very quickly with the number of timers. For this reason we introduce a fixed-point iteration with the aim of solving equations (5.4)-(5.5) numerically.
In Algorithm 2 we show the fixed-point iterations. The algorithm initialises the values for x_k randomly according to independent uniform random variables with support $(0, \gamma_k)$. Recall that the x_k may be interpreted as the reversed rates associated with the death transition in the model of Figure 5.2 and hence cannot exceed γ_k. Then, we iteratively compute ρ_k and x_k according to Equations (5.4) and (5.5), respectively, until we reach the desired accuracy, i.e., the L1-norm of the difference between two successive iterations on ρ_k is lower than ε.
Although we do not have a formal proof of the convergence for Algorithm 2, we carried out several tests with randomly generated models and always observed the computation of a good approximation for ρ_k.

The convergence of Algorithm 2 has been verified on $100,000$ random models with a population ranging from $1,000$ to $100,000$ objects. We have also tested the scalability of the

Algorithm 2: Fixed-point algorithm for the solution of the system of rate equations (5.5).

Data: $\lambda_k, \gamma_k, \eta, \varepsilon$
Result: ρ_k
Initialise randomly x_k for $k = 1, \dots K$ with uniform distribution in $(0, \gamma_k)$;
Compute ρ_k using Equation (5.4);
repeat
  $\rho'_k \leftarrow \rho_k$ for all $k = 1, \dots, K$;
  Update x_k using ρ'_k by Equation(5.5);
  Compute ρ_k using Equation (5.4);
until $|\rho'_k - \rho_k| < \varepsilon$;

fixed point algorithm with different numbers of objects. Using an Intel Core(TM)2 Duo CPU processor, we have analysed a model with 1000 objects and obtained the solution of the non-linear system in $0.5s$. We have noticed that the convergence time depends on the variance of the set of λ_ks. Indeed, for $100,000$ objects, the computation time varies between 30 and 160 seconds, where the lowest time corresponds to the lowest variance.

5.2 Application: analysis of an ideal TTL cache with rejuvenation

In this section we use the results derived in Section 5.1.3 to analyse a class of TTL caches with rejuvenation. TTL caches are attracting the attention of the research community especially in the context of the analysis of Information-Centric Networks [57]. Analyses of TTL cache networks are shown, e.g., in [15, 34, 35]. In TTL caches, each object is associated with a timer. At each object access, the timer is reset and when it expires the corresponding object is evicted from the cache. Here, we propose a rejuvenation mechanism for the timers that reduces their ages in such a way that the last recently used is set to 0. This aims at reducing the problem of under-utilisation of the TTL cache in case of long periods of inactivity. The model we propose here represents an *ideal case* in the sense that we require the maintenance of a timer for all the objects, including those which are not present in the cache. With the model proposed in Section 5.2.4 we will propose a workaround to this implementation issue.

5.2.1 System description

We study the cache model under the Independent Reference Model (IRM) assumptions in continuous time. IRM requires the object requests to be generated according to independent Poisson processes. Although this scenario may be unrealistic for some practical performance analysis, still it is widely used as a benchmark to compare different caching policies as, e.g., in [77, 45, 30, 33, 108]. With reference to the model presented in Section 5.1.3, we have that K is the number of timers associated with the objects, $X_k(t)$ is the state of timer t. Requests of object k occur according to an independent Poisson process with rate λ_k and the partial rejuvenation signal occurs with rate η and starts from an object chosen randomly with uniform distribution. At the k-th object request epoch we observe a cache hit if the timer associated with object k is not greater than threshold $T = T_k$, a cache miss otherwise.

5.2.2 Stationary performance indices

In this section we derive some performance indices of the model in equilibrium. We will give an expression for the cache hit/miss rates and probabilities, and for the expected size of the cache as functions of ρ_k, with $k = 1, \dots K$. Suppose that object k requires α_k bytes to be stored in the cache. Let us define the following stochastic process that corresponds to the cache size:

$$S(t) = \sum_{k=1}^{K} \alpha_k \mathbf{1}_{X_k(t) \leq T}$$

where $\mathbf{1}_{X_k(t) \leq T}$ is 1 if $X_k(t) \leq T$, 0 otherwise. We are interested in the evaluation of the expected cache size in equilibrium, i.e.,

$$\overline{S} = E\left[\lim_{t \to \infty} S(t)\right] .$$

Notice that the limit exists since $\lim_{t\to\infty} X_k(t)$ is the marginal distribution associated with the states of timer k of the ergodic CTMC underlying the model.

Proposition 4 (Expected cache size)**.** In equilibrium, the expected cache size $\overline{S}$ is:

$$\overline{S} = \sum_{k=1}^{K} \alpha_k (1 - \rho_k^{T+1}) \tag{5.7}$$

Proof. The proof follows from the observation that in equilibrium the timer models behave as if they were independent (product-form). Therefore, the probability that the timer k is not over T is:

$$\lim_{t \to \infty} Pr\{X_k(t) \leq T\} = 1 - \rho_k^{T+1} . \tag{5.8}$$

The result is readily derived by weighting the stationary probability of finding an object in the cache by its size.

□

The expression for the standard deviation of the cache size can be used combined with Chebyshev's inequality to derive bounds on the probability that the cache size exceeds a given threshold.

Proposition 5 (Standard deviation of the cache size)**.** The standard deviation of the cache size in equilibrium is:

$$\sigma_S = \sqrt{\sum_{k=1}^{K} \alpha_k^2 \rho_k^{T+1} \left(1 - \rho_k^{T+1}\right)} .$$

Proof. The proof follows after simple algebraic simplifications of the expression:

$$\sigma_S^2 = \sum_{k=1}^{K} \alpha_k^2 (1 - \rho_k^{T+1}) - \sum_{k=1}^{K} \alpha_k^2 (1 - \rho_k^{T+1})^2 .$$

□

Proposition 6 (Hit and miss probability)**.** In equilibrium, the probability of observing a cache hit (h_k) or a cache miss (m_k) event for a request of object k is:

$$h_k = 1 - \rho_k^{T+1}\,, \qquad m_k = \rho_k^{T+1}\,.$$

Proof. According to the PASTA property (see, e.g., [92]), in equilibrium, an event that occurs according to an independent Poisson process sees the stationary distribution. Then, the proof follows the lines of that of Proposition 4. □

The following proposition gives an expression for the rate at which we observe a cache hit or a cache miss event when the model is in equilibrium.

Proposition 7 (Total hit/miss rate)**.** In equilibrium, the total cache hit rate is $H = \sum_{k=1}^{K} h_k\lambda_k$ and the miss rate is $M = \sum_{k=1}^{K} m_k\lambda_k$.

Finally, we give an expression for the probability of a request to generate a cache hit or a cache miss:

Proposition 8 (Total hit/miss probability)**.** In equilibrium, the probability that an object request causes a cache hit event (P_H) or a cache miss event (P_M) is given by the following expressions:

$$\begin{aligned} P_H &= \frac{\sum_{k=1}^{K}\lambda_k(1-\rho_k^{T+1})}{\sum_{k=1}^{K}\lambda_k}\,, \\ P_M &= 1 - P_H = \frac{\sum_{k=1}^{K}\lambda_k\rho_k^{T+1}}{\sum_{k=1}^{K}\lambda_k}\,. \end{aligned}$$

We observe that in a perfectly symmetric system, i.e., when all the objects are requested with the same rate and occupy the same space, the cache hit probability and rate are directly proportional to the expected space dedicated to the cache. This should be not surprising since the choice of evicting one particular object at a given time is arbitrary under the IRM assumption given that all the object requests occur with the same rate (recall that IRM inherits the memoryless property of the exponential distribution).

5.2.3 Experiments

In this section we present some experiments in order to evaluate the performances of the TTL cache with reset signals. Considering 1000 objects (and so 1000 timers) with dimension 1, $T = 20$, $\gamma = \gamma_k = 30$ and homogeneous $\lambda_k = 0.5$ for $1 \leq k \leq K$, we tested the effect of η on the hit probability. In this experiment, η varies from 0.001 to 5.0 with a step of 0.1. In Figure 5.3 we can see how η affects the probability of finding an object in the cache.
We study the influence of the variance of λ_k on the hit probability and the average space occupation of the cache. In Figure 5.4 we use sets of λ_k with growing variance and we notice that the fraction of objects in the cache is lower than the hit probability for higher

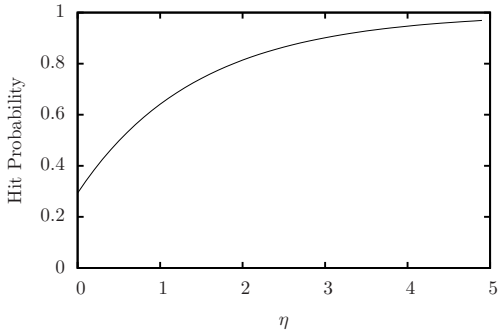


Figure 5.3: Cache hit probability vs. η with homogeneous λ_k.

variances. Intuitively, this happens because in the TTL cache, the objects required with high frequency are present with very high probability and this causes a good cache hit probability even with small cache sizes. As a consequence, the benefits on the cache hit probability with higher values of η are lower.

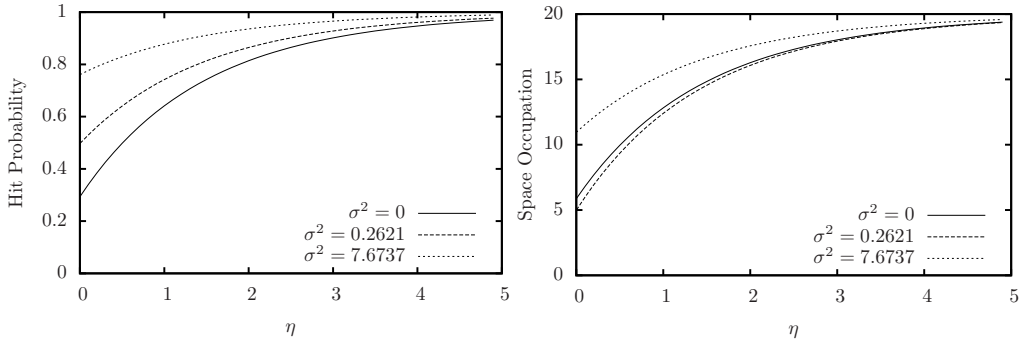


Figure 5.4: Comparison between different heterogeneous λ_k with varying η.

Finally, we study the differences between the TTL model with resets and the model with the FIFO policy studied in [77]. In this case, we took $\gamma = \gamma_k = 80$, the $\lambda_k = (1, \ldots, 1, 10, 10)$ and $\eta = 0.005$. We varied T to get the same space occupation for the two models (for TTL model we consider the average space occupation). With low space occupation, the FIFO model works slightly better but, between 10% and 80% of cached objects, the TTL model with resets shows a higher cache hit probability (up to 10% of improvement) due to the fact that the reset signals avoid the removal of some objects from the cache.

5.2.4 A model for ageing objects with maximum threshold

In this section we consider a model which is similar to that described in Section 5.1.3 but in which the ageing of objects has a maximum threshold. Once this threshold is reached, the object stops its ageing. The motivation to study this type of ageing objects is that when ageing is handled by timers and the proportion of *young* objects is small with respect to the object population, it is too computationally expensive to handle all the timers. So according to this idea we use one state to denote that an object is *very old* and hence we do not handle its timer any more. Clearly, we need to specify how the

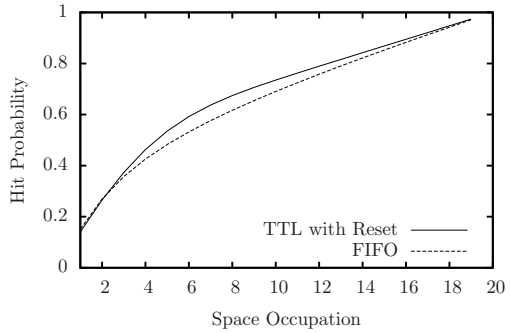


Figure 5.5: Hit probability of TTL cache with reset and FIFO cache.

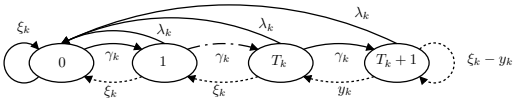


Figure 5.6: Simplified model for an ageing object with maximum age.

partial rejuvenation signal affects the objects in their terminal state. Intuitively, in order to get around the problem of the lack of knowledge of the true age of the objects in their terminal state we introduce a probabilistic behaviour, i.e., object k in its terminal state is partially rejuvenated with a certain probability p_k or it is left in its terminal state with probability $1-p_k$. In both cases, the partial rejuvenation signal is transmitted immediately to object k^+.
In this case the analogue of the simplified model of Figure 5.2 is shown in Figure 5.6. Let $\mathbf{H}(t)$ be the CTMC underlying a set of ageing objects with maximum age and partial rejuvenation and let π^H be its equilibrium distribution function. Moreover, we use T_k+1 to denote the terminal state of ageing object k. When the model is in state T_k+1 and a partial rejuvenation signal arrives it may either jump to state T_k (rate y_k) or stay in state T_k+1 (rate $\xi_k - y_k$), where $y_k < \xi_k$. We now study the conditions on y_k that give the product-form. In order for the model of Figure 5.6 to be quasi-reversible, we must prove that all the dotted transitions have the same reversed rate [61]. A necessary condition to satisfy this constraint is that $\pi_k^H(u)/\pi_k^H(u-1) = \rho_k$, for all $u = 1, \dots T_k$. Let us start by writing down the global balance equation associated with state u, with $0 < u < T_k$:

$$\pi_k^H(u)(\lambda_k + \xi_k + \gamma_k) = \pi_k^H(u+1)\xi_k + \pi_k^H(u-1)\gamma_k \,,$$

which can be rewritten as:

$$\lambda_k + \xi_k + \gamma_k = \rho_k \xi_k + \frac{\gamma_k}{\rho_k} \,. \tag{5.9}$$

Moreover, we have to satisfy the GBEs of states T_k and T_k+1:

$$\begin{cases} \pi_k^H(T_k)(\lambda_k + \xi_k + \gamma_k) = \pi_k^H(T_k+1)y_k + \pi_k^H(T_k-1)\gamma_k \\ \pi_k^H(T_k+1)(y_k + \lambda_k) = \pi_k^H(T_k)\gamma_k \end{cases} \tag{5.10}$$

The last condition we need to satisfy is that the reversed rate of the transitions with rates y_k and $\xi_k - y_k$ must be equal to the reversed rate of the transitions with rate ξ_k, i.e.:

$$\frac{\pi_k^H(T_k+1)}{\pi_k^H(T_k)} y_k = \xi_k - y_k = \rho_k \xi_k \,. \tag{5.11}$$

Notice that we have only one free variable, y_k, to satisfy Equations (5.9), (5.10), (5.11).

Lemma 1. The model depicted in Figure 5.6 is quasi reversible if:

$$y_k = \frac{1}{2}\left(\xi_k - \gamma_k + \sqrt{(\lambda_k + \gamma_k + \xi_k)^2 - 4\xi_k\gamma_k}\right) . \tag{5.12}$$

In this case we have the following equilibrium distribution:

$$\pi_k^H(u) = \begin{cases} \pi_k^H(0)\rho_k^u & \text{if } u \leq T_k \\ \pi_k^H(0)\rho_j^{T_k}\frac{\gamma_k}{\lambda_k+y_k} & \text{if } u = T_k + 1 \end{cases} \tag{5.13}$$

with

$$\rho_k = \frac{\lambda_k + \gamma_k + \xi_k - \sqrt{(\lambda_k + \gamma_k + \xi_k)^2 - 4\xi_k\lambda_k}}{2\xi_k} \tag{5.14}$$

and:

$$\pi_k^H(0) = (1 - \rho_k) .$$

Proof. First we derive ρ_k from Equation (5.9) which gives:

$$\rho_k = \frac{\lambda_k + \gamma_k + \xi_k \pm \sqrt{(\lambda_k + \gamma_k + \xi_k)^2 - 4\xi_k\lambda_k}}{2\xi_k} .$$

We choose the solution given by Equation (5.14) because it is the only one that admits a positive solution for y_k.
From the second equation of System (5.10) we derive $\pi_k^H(T_k + 1)/\pi_k^H(T_k) = \gamma_k/(\lambda_k + y_k)$. If we divide both sides of the first equation for $\pi^H(T_k)$ and substitute the expression of $\pi_k^H(T_k + 1)/\pi_k^H(T_k) = \gamma_k/(\lambda_k + y_k)$, we obtain:

$$\lambda_k + \xi_k + \gamma_k = \frac{\gamma_k y_k}{\lambda_k + y_k} + \frac{\gamma_k}{\rho_k} , \tag{5.15}$$

that can be reduced to a linear equation in y_k whose solution is given by Expression (5.12). We can prove that y_k is positive when all the other rates are also positive. At this point we have proved that for this choice of y_k we have:

- $\pi_k^H(u)/\pi_k^H(u-1) = \rho_k$ for all $0 < u \leq T_k$
- $\pi_k^H(T_k + 1)/\pi_k^H(T_k) = \gamma_k/(\lambda_k + y_k)$

and hence the equilibrium distribution of Equation (5.13) can be readily derived. However, in order to prove that the model is quasi-reversible we still have to check the reversed rates of the dotted transitions of Figure 5.6. Notice that the reversed rates of the transitions with forward rate ξ_k from state u to $u-1$ is $\rho_k\xi_k$. This is equal to the reversed rate of the transition with forward rate y_k which is $\gamma_k y_k/(y_k + \lambda_k)$ as can be seen by comparing Equation (5.15) and the GBE (5.9). It remains to prove that $\xi_k - y_k$ is positive and that its reversed rate, which is equal to its forward rate, is $\rho_k\xi_k$. The verification of this equality is purely algebraic. Finally, we derive the expression of $\pi_k^H(0)$ by normalising the stationary probabilities, i.e.:

$$\sum_{u=0}^{T_k} \pi_k^H(0)\rho_k^u + \pi_k^H(0)\rho_k^{T_k}\frac{\gamma_k}{\lambda_k + y_k} = 1 .$$

After some algebraic reductions one obtains:

$$\pi_k^H(0) = \left[\frac{1-\rho_k^{T_k+1}}{1-\rho_k} + \frac{\rho_k^{T_k}\gamma_k}{\lambda_k + y_k}\right]^{-1} = (1-\rho_k)\,,$$

as required. This concludes the proof.

□

Theorem 5.2 gives the product-form equilibrium distribution for this model. Let $\mathbf{H}(t)$ be the CTMC underlying a set of ageing objects with maximum age and partial rejuvenation.

Theorem 5.2. *The stochastic process $\mathbf{H}(t)$ in which the rates satisfy the condition of Lemma 1 has the product-form equilibrium distribution:*

$$\pi^H(\mathbf{u}) = \prod_{k=1}^{K} \pi_k^H(u_k) \tag{5.16}$$

where ρ_k is the solution of the non-linear system of equations (5.4)-(5.5) and π_k^H is given by Equation (5.13) where $\xi_k = x_k + \eta/K$.

Proof. The proof follows the same steps as that of Theorem 5.1 given Lemma 1 stating the quasi-reversibility of the model in Figure 5.6.

□

Corollary 1 (Partial rejuvenation probability)**.** The model has a product-form if at the arrival of a partial rejuvenation signal at an object in state T_k+1, the probability of changing its state to T_k is:

$$p_k = \frac{\xi_k - \gamma_k - \lambda_k + \sqrt{(\gamma_k + \lambda_k + \xi_k)^2 + 4\gamma_k\xi_k}}{2\xi_k}\,, \tag{5.17}$$

where $\xi_k = x_k + \eta/K$.

Proof. The expression is obtained by simply computing y_k/ξ_k, since ξ_k is the rate at which the partial rejuvenation signal arrives at the object.

□

The following result is important for understanding the connections between the model studied in Section 5.1.3 with unbounded ageing and the one considered here. The result is in some sense surprising since it basically states that with the definition of rate y_k as specified by Equation (5.12) we have two enjoyable properties: the first is that the equilibrium distribution is in product-form and hence analytically tractable, and the second is that the stationary probability of observing a state u, with $1 \leq u \leq T_k$ is the same in the model with unbounded ageing and that with bounded ageing. Therefore, in a practical situation in which the object ageings are handled by timers, one can replace the ideal model with unbounded ageing with the one proposed in this section while maintaining the same performance indices, provided that the probability of rejuvenating an object k in state $T_k + 1$ upon the arrival of a partial rejuvenation signal is set according to Equation (5.17).

Corollary 2 (Equivalence corollary)**.** Given the CTMCs $\mathbf{X}(t)$ and $\mathbf{H}(t)$ and let π^X and π^H be their equilibrium distribution functions. Then

$$\pi^H(\mathbf{u}) = \prod_{i=1}^{K} g_k(u_k)\,,$$

where $\mathbf{u} = (u_1, \ldots, u_K)$ and

$$g_k(u_k) = \begin{cases} \pi_k^X(u_k) & \text{if } 0 \leq u_k \leq T_k \\ \sum_{u'=T_k+1}^{\infty} \pi_k^X(u') & \text{if } u_k = T_k + 1 \end{cases}$$

Therefore, we can see $\mathbf{H}(t)$ as an aggregation of $\mathbf{X}(t)$ that preserves the equilibrium distribution and the product-form.

It is worth note that $\mathbf{H}(t)$ is not a lumping (neither strong nor exact [71, 87]) of $\mathbf{X}(t)$. Moreover, we observe that the choice of y_k can be interpreted as the conditional transition rate from state $T_k + 1$ to state T_k of object k considered in isolation:

$$y_k = \frac{\pi_k^X(T_k + 1)}{\sum_{u'=T_k+1}^{\infty} \pi_k^X(u')} \xi_k\,.$$

5.2.5 Application: Revisiting the TTL cache with partial rejuvenation

In Section 5.2 we have shown an application of our theoretical findings in the analysis of a single TTL cache with partial rejuvenation. We called the model *ideal* since its actual implementation would be prohibitive because it would requires to associate a timer with each object, including those which are not in the cache. However, we overcome this problem by using the model presented in this section. For each object k we have a timer that is handled only when the object is in the cache, i.e., its state u_k is $0 \leq u_k \leq T_k$. When the object is evicted from the cache, its state is $T_k + 1$. The partial rejuvenation signal has a probabilistic effect on the objects outside the cache, i.e., they are rejuvenated with probability p_k as specified by Equation (5.17) or they stay in state $T_k + 1$ with probability $1 - p_k$. In both cases the object instantaneously propagates the signal to the following object k^+. By Corollary 2 the performance measures are the same as derived in Section 5.2. We now address the inverse problem, i.e., finding γ_k and T_k such that the probability of copying in the cache an object which is outside, i.e., in state $T_k + 1$, is q_k and the expected eviction time is R_k. From a theoretical point of view, the problem has a simple solution:

$$\gamma_k = \frac{(1 - q_k)(\lambda + q_k \xi_k)}{q_k}\,, \tag{5.18}$$

then we may choose T_k in such a way that the expected eviction time is R_k, i.e., $\lceil T_k = \gamma_k R_k \rceil$. Nevertheless, the evaluation of Equation (5.18) requires the solution of the non-linear system (5.4)-(5.6) which may be computationally expensive to be performed dynamically. A dynamic adjustment of γ_k in order to obtain the desired probability q_k can be formulated by observing that p_k in monotonic on γ_k in Equation (5.17). Indeed, we have:

$$\frac{dp_k}{d\gamma_k} = \frac{1}{2\xi}\left(-1 + \frac{\gamma_k + \lambda_k - \xi_k}{\sqrt{(\gamma_k + \lambda_k + \xi_k)^2 - 4\gamma_k \xi_k}}\right)$$

which is always negative for positive transition rates. In other words, the dynamical control of p_k in order to reach the desired value q_k can be performed by simply augmenting γ_k (and reducing T_k) if the current value of p_k is higher than q_k, and by reducing it when it is lower.

5.3 Conclusion

With our work, we propose two novel product-form models that can be applied for the analysis of systems with ageing objects. These two models differ because in one the objects have an unbounded age while in the other there is a maximum threshold for the age. The approach that we propose can be applied also for the analysis of heterogeneous systems, i.e., systems in which some objects have a maximum threshold for the age and some others have not.
Informally, we can say that the peculiarity of these models is that the transitions are not "local" -as in most of product-form models- i.e., they may change the states of all the objects instantaneously and the effect of a partial rejuvenation event depends on the global state of the model.
We show that the CTMC underlying the model of objects with limited ageing can be seen as an aggregation of the chain of the model with unbounded ageing. This allowed us to prove some equivalence results on the expected performance indices. The consequences of these equivalences are important for practical applications especially if the system must maintain a timer for the objects. Indeed, by introducing a maximum age, we avoid maintaining the timers of all the objects that reach this age thus reducing the computational effort required for monitoring the object ageing. We used as case-study the analysis of a cache with TTL policy and partial rejuvenation of the objects.
The results can be extended in order to include the partial rejuvenation of clusters of objects (instead of all) and more sophisticated interactions among them in a similar fashion of what is considered in [41]. Indeed, our model may be seen as a queueing network with external independent Poisson arrival streams. The queues can only be partially flushed (i.e., the customers in the network are reduced as computed by Algorithm 1) or totally flushed, i.e., the number of customers in a single queue is set to 0. The exponential distribution of the equilibrium distributions stated by Theorem 5.1 allows the introduction of state-independent probabilistic customer routing.
The proposed model could also be further developed with the possibility of a probabilistic insertion of an object in the cache, i.e., when an object is not in the cache and it is requested, it can enter the cache with a probability q or remain outside with probability $1 - q$. Moreover, we aim at overcoming the limitations of the IRM by allowing state-dependent request rates so that the last requested objects will be associated with a higher rate thus incorporating in the analysis the time-locality property of the network traffic.

Chapter 6

Analysis of reversible computations

6.1 Introduction

Reversible computations have been widely studied from the point of view of functional and energy consumption. In the literature, several authors have proposed various formalisms (mainly based on process algebras) for assessing the correctness or the equivalence between reversible computations. In this chapter we propose the adoption of Markovian stochastic models to assess the quantitative properties of reversible computations. Under some conditions, we show that the notion of time reversibility for Markov chains can be used to efficiently derive some performance measures of reversible computations. The importance of time-reversibility relies on the fact that, in general, the process's stationary distribution can be derived efficiently by using numerically stable algorithms. We will reviews the main results about time-reversible Markov processes and discusses how to apply them to tackle the problem of the quantitative evaluation of reversible computations.

6.2 Motivations

Reversible computations have two execution directions: forward, corresponding to the usual notion of computation, and backward that restores previous states of the execution. Various applications and problems related to reversible computations have been widely studied in different research areas and from different viewpoints, including functional analysis and energy consumption (e.g., [80, 96] and the references therein). Various formalisms and models have been proposed in the literature to represent and assess qualitative properties of reversible computations such as their correctness or if two reversible processes are equivalent in some terms. Most of the proposed approaches are based on process algebras that do not include any notion of computation time [31, 80]. We focus on the quantitative analysis and evaluation of reversible computations based on Markov stochastic processes. The dynamic behaviour of the forward and backward computation may be represented by stochastic models that include the notion of time. Hence, under certain conditions, time-reversibility of stochastic processes can be applied to assess quantitative properties of reversible computations.
Quantitative models based on Markov processes have been widely applied for the analysis and evaluation of complex systems (see e.g., [44, 21]). Markov models and formalisms have the advantage of efficient methods and algorithms for studying their behaviour. In

particular, under appropriate stationary conditions one can derive the equilibrium state distribution of a continuous-time Markov chain by applying algorithms with polynomial time complexity in the process state space cardinality [103]. Several higher level formalisms that are widely applied for quantitative analysis are based on Markov processes, including Stochastic Process Algebras (SPA), Stochastic Petri Nets (SPN), Stochastic Automata Networks (SAN) and Queueing Networks (QN). Although the quantitative analysis based on these formalisms can be obtained by the direct solution of the underlying Markov chain, the state space dimension of the process in general grows exponentially with the model dimension. This is known as the state-space explosion problem and becomes intractable from the computational viewpoint as the problem size increases. In order to overcome this problem, various techniques have been proposed in the literature, including the state-space reduction by aggregating (or lumping) methods, approximation techniques, and the identification of product-form solutions for state probabilities of the Markov chain. The product-form theory provides techniques to derive the equilibrium state distribution of a complex model based on the analysis of its components in isolation. As we said in Section 3.5, product-form models consist of a set of interacting sub-models whose solutions are obtained by isolating them from the rest of the systems. Then, the stationary state distribution of the entire model is computed as the (normalised) product of the stationary state distributions of the sub-models. Various classes of product-form models have been defined for different formalisms and some of them can be analysed through efficient algorithms with a low polynomial complexity in the model dimension. Product-form has been widely investigated for queueing network models [14, 74]. These product-form models have simple closed-form expressions of the stationary state distributions that lead to efficient solution algorithms. For more general Markov models and by the compositionality property of Stochastic Process Algebra, the Reversed Compound Agent Theorem (RCAT) [60, 11] provides a product-form solution of a stationary CTMC defined as a cooperation between two sub-processes under certain conditions, as we saw in Section 3.7.3. This result gives a unified view of most of the commonly used product-forms.

The concept of time-reversibility of Markov stochastic processes has been introduced and applied to the analysis of Markov processes and stochastic networks by Kelly [76]. A reversible Markov process has the property that when the process obtained by reversing the direction of time is reversed has the same probabilistic behaviour as the original one. Early applications of these results lead to the characterisation of product-form solutions for some models with underlying time-reversible Markov processes, such as closed exponential Queueing Networks [14, 58]. Also the RCAT characterisation of product-form solutions is connected to time-reversibility: the solution is based on the definition of a set of transition rates in the time-reversed process. Further notions of reversibility have been introduced in [110, 76] for dynamically reversible processes where some states of the direct and reversed processes are interchanged, and more recently the ρ-reversibility for reversible processes with arbitrary state renaming [87, 86]. Some results on properties and product-form solutions have been recently derived for this class of time-reversibility [89]. We will survey the main results about time-reversible Markov processes and discuss how to apply them to address the problem of quantitative evaluation of reversible computations. We recall the definition of time reversibility for continuous time Markov processes, the main properties and its application for quantitative analysis. We present an abstract model of continuous time Markov chain for representing and performance evaluating reversible parallel computations. Taking advantage of the process reversibility,

the stationary distribution of the model can be efficiently derived by using numerically stable algorithms. In particular we present some product-form results of reversible synchronising automata by applying ρ-reversibility to the underlying Markov process.

6.3 Model and analysis

6.3.1 Time reversibility for CTMCs (ρ-reversibility)

Given a stationary CTMC, $X(t)$ with $t \in \mathbb{R}$, we call $X(\tau - t)$ its reversed process for all $\tau \in \mathbb{R}$. We denote by $X^R(t)$ the reversed process of $X(t)$. It can be shown that $X^R(t)$ is also a stationary CTMC. Given a state renaming function ρ (a bijection from S to S), we say that $X(t)$ is ρ-reversible if it is stochastically identical to $X^R(t)$ modulo the state renaming ρ [86, 87]. Intuitively, an external observer is not able to distinguish $X(t)$ from $X^R(t)$ once the state renaming function ρ is applied to rename the states. Notice that if ρ is the identity then we simply say that $X(t)$ is *reversible*, whereas if ρ is an involution, then we say that $X(t)$ is *dynamically reversible* [110, 76].
We can decide if a CTMC is ρ-reversible in two ways:

- the first involves the steady-state distribution of the CTMC;
- the latter is based on an extended formulation of Kolmogorov's criteria [76], i.e., requires the analysis of the cycles in the reachability graph.

Lemma 2. Given a stationary CTMC $X(t)$ with state space S, if there exists a collection of positive real numbers π summing to unity and a bijection ρ from S to S such that:

$$q_s = q_{\rho(s)} \qquad \forall s \in S \tag{6.1}$$

$$\pi(s)q(s,s') = \pi(\rho(s'))q(\rho(s'),\rho(s)) \quad \forall s,s' \in S, s \neq s' \tag{6.2}$$

then $X(t)$ is ρ-reversible and $\pi(s)$ is its steady-state distribution.

Equation 6.1 states that the residence time of a state and its renaming must be equal. Notice that this condition is trivially satisfied if ρ is the identity, i.e., $X(t)$ is reversible. The set of equations 6.2 are called detailed balance equations. In case the renaming function ρ is known, it is possible to use the detailed balance equations to compute the chain's steady-state distribution instead of the more complex GBE.

Lemma 3. Given a stationary CTMC $X(t)$ with state space S and let ρ be a renaming on S. $X(t)$ is ρ-reversible with respect to ρ if and only if for every finite sequence $s1, s2, ..., s_n \in S$:

$$\begin{aligned} q(s_1,s_2)q(s_2,s_3)q(s_{n-1},s_n)q(s_n,s_1) = \\ q(\rho(s_1),\rho(s_n))q(\rho(s_n),\rho(s_{n-1}))q(\rho(s_3),\rho(s_2))q(\rho(s_2),\rho(s_1)) \end{aligned} \tag{6.3}$$

and Equation 6.1 holds $\forall s \in S$.

6.3.2 Modelling reversible computations with ρ-reversible Markov processes

Reversible computations are characterised by the fact that they have two execution directions: the forward and the backward that restores past states of the computation. Our idea of the implementation of purely reversible computations is similar to that considered in [96], i.e., the code being executed is naturally reversible. By purely reversible

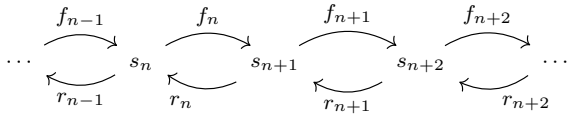


Figure 6.1: Model for a reversible sequential computation.

computations we mean those computations in which each step can be undone and there are no segments in which the execution direction is forward only. For instance, the programmer may have used Janus [111] which is a programming language for reversible computations or a subset of a standard language equipped with a reversible compiler.

6.3.2.1 Modelling reversible programming structures

In this section we describe a modelling methodology for the reversible programming structures such as sequences, branches, cycles and sequences with checkpoints.

Sequential computations. The simplest reversible computation is the reversible sequential one shown in Figure 6.1 where s_i are the states of the computation and the arc labels denote the transition rates, f standing for forward rates and r for the reversed ones. In this model every state can be restored in one step. For each state we define a probabilistic law that decides if the computation will proceed in the forward or backward direction. In practice these probabilities can be derived by the statistical analysis of the software execution or by the knowledge of the intrinsic law that governs the probability of proceeding in one direction or the opposite.
Assume that the residence time in state s_n is exponentially distributed with rate f_n+r_{n-1}, then the probability of a forward transition given that $X(t)$ (i.e. the Continuous Time Markov Chain) is in state s_n is $f_n/(f_n + r_{n-1})$ and the probability of a backward transition is $r_{n-1}/(f_n + r_{n-1})$. This follows from the properties of the exponential random variable (see, e.g., [98]) and the so called *race policy*.
If the Markov chain depicted in Figure 6.1 is ergodic then it is reversible. The ergodicity is trivially satisfied if there exist lower and higher boundary states. The former is a state that does not allow a backward computation while the latter is a state that does not allow a forward computation. According to Lemma 3 we have that the forward cycle $s_n \xrightarrow{f_n} s_{n+1} \xrightarrow{r_n} s_n$ has itself as inverse cycle and therefore the conditions of Lemma 3 are satisfied.

Branches. Branches can be modelled in a similar way to the one used for the sequential computations. Notice that, as commonly done in stochastic modelling, we model the branch by means of the probabilistic behaviour of the executed process. Although a modelling approach taking into account the detailed description of the system state is theoretically possible, in many cases this is not practically feasible due to the high cardinality that would be reached by the state space.
The forward rates f is split among all the possible branches that we have to reach. Suppose that the system is going forward and state s_1 is associated with a branch that proceeds to state s_2 with probability p and to s_3 with probability $1 - p$ (see Figure 6.2). In this case, let $1/f$ be the expected residence time in state s_1, then the transition rates are $f_1 = fp$ and $f_2 = f(1 - p)$. Following the reasoning proposed in the previous paragraph on sequential computations, it is easy to see that the conditions of Lemma 3 are satisfied

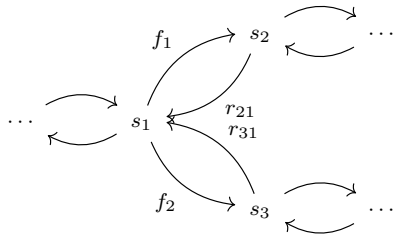


Figure 6.2: Model for a reversible branch.

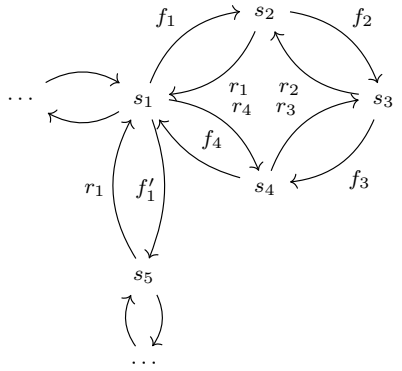


Figure 6.3: Model for a reversible cycle.

by choosing ρ as the identity.

Cycles. Cycles can be modelled as long as each transition they consist of can be undone. Let us consider the model of Figure 6.3, where we can choose more than one possible cycle but for simplicity we focus just on the cycle s_1, s_2, s_3, s_4 as the forward direction and s_1, s_4, s_3, s_2 as the reverse one (although similar reasoning can be done choosing other cycles). The computation at state s_1 can proceed by entering the cycle s_1, s_2, s_3, s_4 or by moving to state s_5. The probability of entering the cycle given that the computation will proceed in the forward direction is $f_1/(f_1 + f_1')$ and the number of (forward) iterations are geometrically distributed. Modelling the exact number of iterations of the cycles is possible but, in general, will drastically increase the number of model states. Let us focus on the cycle s_1, s_2, s_3, s_4 and its inverse s_1, s_4, s_3, s_2. If we apply Lemma 3 with ρ being the identity function, we notice that the conditions are satisfied for the cycles consisting of two states (e.g., s_3, s_4, s_3) but we need also to consider the cycle s_1, s_2, s_3, s_4 whose inverse is s_1, s_4, s_3, s_2 that originates a rate-condition for the ρ-reversibility: $f_1 f_2 f_3 f_4 = r_1 r_2 r_3 r_4$. In general, in cycles, the product of the forward rates must be equal to the product of the corresponding backward rates. This is trivially satisfied if the time required to perform a forward computation follows the same distribution of that required to undo it.

Sequences with checkpoints. In the previous paragraphs we have shown how it is possible to model reversible sequential computations, branches and cycles by using a reversible CTMC, i.e., by taking the identity as ρ function. In the context of modelling reversible computations, the notion of ρ-reversibility is important because it allows the specification of atomic sequences that can be only fully reversed. For instance, consider

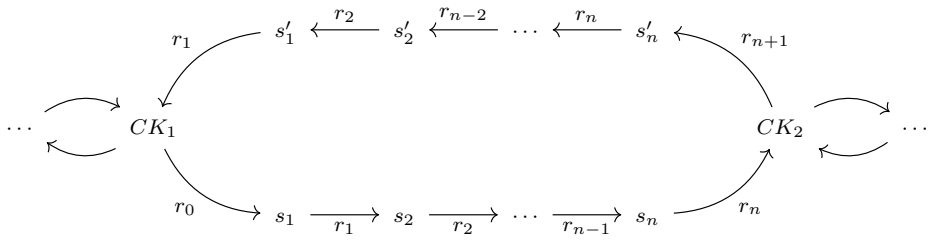


Figure 6.4: Model for a reversible computation with checkpoints.

a system atomic transaction whose correctness is tested at a certain checkpoint. If an invalid state is detected, then all the operations performed by the transaction must be undone. An example of such a computation is shown in Figure 6.4. In order to prove the ρ-reversibility of the model, we define function ρ as:

$$\rho(s) \begin{cases} CK_1 & \text{if } s = CK_1 \\ CK_2 & \text{if } s = CK_2 \\ s'_i & \text{if } s = s_i \;\; 1 \leq i \leq n \\ s_i & \text{if } s = s'_i \;\; 1 \leq i \leq n \end{cases}$$

By Lemma 3 we observe that the residence time of s_i must have the same expectation of that of its ρ-renaming, s'_i (and vice versa). Therefore, we have the rate condition for the ρ-reversibility whose interpretation is that the time required to perform an operation in the transaction must follow the same probabilistic law of that required to undo it. For what concerns the cycle analysis, observe that $CK_1, s_1, ..., s_n, CK_2, s'_n, ..., s'_1, CK_1$ has itself as inverse and hence the condition 6.3 is satisfied.

6.3.2.2 Modelling assumptions and steady-state

In this section we discuss two crucial points of the modelling technique that we propose:

- How does the exponential assumption of the distribution of the state residence time impact on the expressiveness of this modelling framework?
- How do we interpret the steady-state distribution of Markov chains in terms of quantitative properties of the reversible computations?

The exponential assumption can be relaxed by using distributions whose coefficient of variation may be higher or lower than that of the exponential. This is achieved by splitting a state whose residence time is not exponential into a set of micro-states each of which has an exponential residence time. Coxian random variables are formed by exponential stages and can approximate any distribution with rational Laplace transform with arbitrary accuracy (see, e.g.,[75]). The literature proposing algorithms to fit data statistics to a distribution by means of a combination of the exponential stages is very rich (e.g., [23, 93]). Informally, the steady-state distribution of a CTMC is the probability of observing a given state when the time elapsed since the first observation is very large (the time required to reach the stationary behaviour depends on the magnitude of the second eigenvalue of the infinitesimal generator). For instance, in stationary reversible simulations [96] the state of the process after a period of warm up, is independent of its initial conditions and hence our framework can be applied easily. The assumption that

each state transition can be undone includes the transitions that take the model to the state encoding the result of the computation. As a consequence, it is not obvious how the steady-state distribution can give an idea about the time required to obtain the result in a reversible computation. In stochastic analysis this problem is connected to the computation of the (moments of the) distribution of the time to absorption. Basically, we assume that once the chain enters in one of the states encoding the result, then they cannot leave them. Unfortunately, to the best of our knowledge, time-reversibility does not help in the exact computation of the distribution of the time to absorption. Nevertheless, approximating methods which may take advantage from the process' reversibility are available and are quite accurate when the expected computation time is much higher than the expected transition delays of the model (see, e.g., [12, 4]). The steady-state distribution may also be interpreted as the fraction of a large number of processes which are in a given state (in the long-run) once they are run in parallel and they restart their computation after terminating it.

6.3.3 Cooperation of reversible parallel computations

In section 3.3 we presented Stochastic Automaton (SA) as an abstract model based on continuous time Markov chains, this model can be used for the performance evaluation of reversible parallel computations. Differently from those functional models that represent explicitly the parallel composition of reversible computations, we do not consider any notion of causality. Instead we can use SA for analysing the dynamic behaviour of those computations that can be realized in a reversible fashion, where the underlying conditional probabilities play the role of causality.

Notice that, according to the definition of SA, an automaton obtained by a composition does not have passive types. This is reasonable if we consider the fact that in this case the resulting automaton has an underlying CTMC and then we can study its equilibrium distribution. In [89] we can see that this semantics for pairwise SA synchronisations can be easily extended in order to include an arbitrary finite number of pairwise cooperating automata.

6.3.3.1 Reversible Stochastic Automata

We now introduce the notion of ρ-reversibility for stochastic automata. We present a definition in the style of the Kolmogorov's criteria stated in [76]. We assume the existence of a bijection (renaming) $\overleftarrow{\cdot}$ from $\mathcal{T}_P$ to $\mathcal{T}_P$ such that for each forward action type a there is a corresponding backward action type $\overleftarrow{a}$ with $\overleftarrow{\tau} = \tau$. In most of practical cases, $\overleftarrow{\cdot}$ is an involution, i.e., $\overleftarrow{\overleftarrow{a}} = a$ for all $a \in \mathcal{T}_P$, and hence the semantics becomes similar to the one proposed in [31]. We say that $\overleftarrow{\cdot}$ respects the active/passive types of an automaton P if $\overleftarrow{\tau} = \tau$ and for all $a \in \mathcal{T}_P \setminus \{\tau\}$ we have that $a \in \mathcal{A}ct_P \Leftrightarrow \overleftarrow{a} \in \mathcal{A}ct_P$, (or equivalently $a \in \mathcal{P}ass_P \Leftrightarrow \overleftarrow{a} \in \mathcal{P}ass_P$).
The notion of ρ-reversible automaton is defined as follows.

Definition 6.1. (ρ-reversible automaton) Let P be an irreducible stochastic automaton. Assume that:

- $\rho : S_P \leftarrow S_P$ is a renaming (permutation) of the states;
- $\overleftarrow{\cdot}$ is a bijection from $\mathcal{T}_P$ to $\mathcal{T}_P$ that respects the active/passive typing.

We say that P is ρ-reversible if:

1. $q(s,a) = q(\rho(s),a)$, for each state $s \in S_P$;
2. for each cycle $\phi = (s_1 \overset{a_1}{\rightsquigarrow} s_2 \overset{a_2}{\rightsquigarrow} ... \overset{a_{n-1}}{\rightsquigarrow} s_n \overset{a_n}{\rightsquigarrow} s_1)$ in P there exists one cycle $\overset{\leftarrow}{\phi} = (\rho(s_1) \overset{\overset{\leftarrow}{a}_n}{\rightsquigarrow} \rho(s_n) \overset{\overset{\leftarrow}{a}_{n-1}}{\rightsquigarrow} ... \overset{\overset{\leftarrow}{a}_2}{\rightsquigarrow} \rho(s_n) \overset{\overset{\leftarrow}{a}_1}{\rightsquigarrow} \rho(s_1))$ in P such that:

$$\prod_{i=1}^{n} q(s_i, s_{i+1}, a_i) = \prod_{i=1}^{n} q(\rho(s_{i+1}), \rho(s_i), \overset{\leftarrow}{a}_i) \text{ with } s_{n+1} \equiv s_1$$

We say that $\overset{\leftarrow}{\phi}$ is the inverse of cycle ϕ. If ρ is the identity function we simply say that P is reversible.

Notice that the inverse cycle $\overset{\leftarrow}{\phi}$ of a cycle ϕ is unique. This can be easily derived from the fact that, by definition of stochastic automaton, there exists at most one transition between any pair of states with a certain type $a \in \mathcal{T}_P$. The following theorem states that any ρ-reversible automaton satisfies a set of detailed balance equations similar to those presented in Lemma 2.

Theorem 6.2. *(Detailed balance equations) If P is ergodic and ρ-reversible then for each pair of states $s, s' \in S_P$, and for each type $a \in \mathcal{T}_P$, we have:*

$$\pi_P(s) q(s, s', a) = \pi_P(s') q(\rho(s'), \rho(s), \overset{\leftarrow}{a})$$

The next proposition says that the states of an ergodic ρ-reversible automaton have the same equilibrium probability of the corresponding image under ρ.

Proposition 9. (Equilibrium probability of the renaming of a state) If P is an ergodic and ρ-reversible automaton then for all $s \in S_P$:

$$\pi_P(s) = \pi_P(\rho(s))$$

6.3.4 Product-form result

It is well-known that the cardinality of the state space of complex systems can grow exponentially with the structure of the model. Even worse, the numerical algorithms for deriving the equilibrium distribution become numerically unstable and prohibitive in terms of computation time. In this section we present the product-form result for networks of ρ-reversible synchronising automata. First we prove that the parallel composition of ρ-reversible automata is still ρ-reversible. Then, based on this result, we prove that the equilibrium distribution of the composition of two ρ-reversible automata can be derived from the equilibrium distribution of the cooperating automata considered in isolation (i.e., without generating the joint state space and solving the system of global balance equations). The analysis in isolation requires us to set a rate for the passive transitions. To this aim, in [89] we can see that, thanks to the rescaling property of ρ-reversible automata, we can choose an arbitrary positive constant.

Theorem 6.3. *(Closure under ρ-reversibility) Let P_1 and P_2 be two ρ_1- and ρ_2-reversible automata with respect to the same function $\overset{\leftarrow}{\cdot}$ on the action types. Then, the composition $P_1 \bigotimes P_2$ is ρ-reversible with respect to the same $\overset{\leftarrow}{\cdot}$, where, for all $(s_1, s_2) \in S_{P_1} \times S_{P_2}$:*

$$\rho(s_1, s_2) = (\rho_1(s_1), \rho_2(s_2))$$

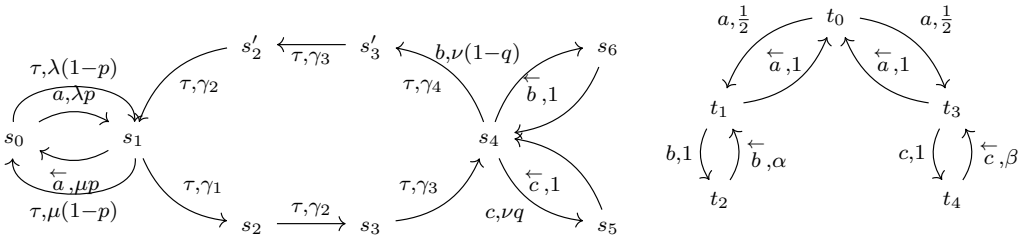


Figure 6.5: Models for P_1 (*left*) and P_2 (*right*)

The next theorem provides the product-form result for networks of ρ-reversible stochastic automata. In order to understand the relevance of this result, consider a set of M cooperating automata and assume that each automaton has a finite state space of cardinality N. The state space of the network may have the size of the Cartesian product of the state space of each single automaton, i.e., in the worst case, its cardinality is N^M. Since the computation of the equilibrium distribution of a CTMC requires the solution of the linear system of global balance equations, its complexity is $O(N^{3M})$. For ρ-reversible automata, by applying Theorem 6.2, we can efficiently compute the equilibrium distribution of each automaton in linear time on the cardinality of the state space, and by Theorem 6.4 the complexity of the computation of the joint equilibrium distribution is $O(NM)$.

Theorem 6.4. *(Product-form solution) Let P_1 and P_2 be two ergodic ρ_1- and ρ_2-reversible automata with respect to the same function $\overleftarrow{\cdot}$ on the action types, and let π_1 and π_2 be the equilibrium distributions of the CTMCs underlying P_1 and P_2, respectively. If $P_1 \bigoplus P_2$ is ergodic on the state space given by the Cartesian product of the state spaces of P_1 and P_2, then for all $(s_1, s_2) \in S_{P_1} \times S_{P_2}$:*

$$\pi(s_1, s_2) = \pi_1(s_1)\pi_2(s_2)$$

where π is the equilibrium distribution of the CTMC underlying $P_1 \bigotimes P_2$. In this case we say that the composed automaton exhibits a product-form solution.

6.4 Discussion

Notice that this analysis, differently from those based on the concepts of quasi-reversibility [76, 61] and reversibility, does not require a re-parameterisation of the cooperating automata, i.e., the expressions of the equilibrium distributions of the isolated automata are as if their behaviours are stochastically independent although they are clearly not.

6.4.1 Example

In this section we describe a model for the parallel composition of two reversible computations. Consider the stochastic automata P_1 and P_2 depicted in Figure 6.5. P_1 and P_2 communicate on the reversible channels a, b and c. Channel a is unreliable, i.e., a packet sent from P_1 to P_2 is received by P_2 with probability p and lost with probability $1 - p$. P executes its computations in the forward ($s_0 \to s_1 \to s_2 \to s_3 \to s_4 \to s_5$ or $s_0 \to s_1 \to s_2 \to s_3 \to s_4 \to s_6$) or backward ($s_5 \to s_4 \to s'_3 \to s'_2 \to s_1 \to s_0$ or $s_6 \to s_4 \to s'_3 \to s'_2 \to s_1 \to s_0$) direction. It has two checkpoints modelled by states s_1 and s_4. P_2 moves from t_0 to t_1 or t_2 with a probabilistic choice upon the synchronisation

with type a. P_1 is ρ_1-reversible with $\rho(s_i) = s_i$ for $i = 0, 1, 4, 5, 6$ and $\rho_1(s_i) = s'_i$ and $\rho_1(s'_i) = s_i$ for $i = 2, 3$, while P_2 is ρ_2-reversible where ρ_2 is the identity function. Notice that $a, \overleftarrow{a}, b, c \in \mathcal{A}ct_{P_1} = \mathcal{P}ass_{P_2}$ and $\overleftarrow{b}, \overleftarrow{c} \in \mathcal{A}ct_{P_2} = \mathcal{P}ass_{P_1}$.
We assume that the model encodes the result of the computation in the states $(s_5, t_2), (s_5, t_4)$, $(s_6, t_2), (s_6, t_4)$. We aim to compute the equilibrium probability of these four states that represents the fraction of time that the process spends in the states that encode the desired result. Now we use Theorem 6.2 to derive the equilibrium distribution of the isolated automata. Let us consider an arbitrary state in P, say s_0. We can immediately derive $\pi_1(s_1)$ by using the detail balance equation and we obtain:

$$\pi_1(s_0)\lambda(1-p) = \pi_1(s_1)\mu(1-p)$$

which gives $\pi_1(s_1) = \pi_1(s_0)\lambda/\mu$.
Then, we derive $\pi_1(s_2)$ using the detailed balance equation with s_1 and obtain:

$$\pi_1(s_2) = \pi_1(s_0)\lambda\gamma_1/(\mu\gamma_2)$$

By Proposition 9 we immediately have $\pi_1(s'_2) = \pi_1(s_2)$.
Then we derive:

$$\pi_1(s'_3) = \pi_1(s_3) = \pi_1(s_0)\lambda\gamma_1/(\mu\gamma_3)$$

$$\pi_1(s_4) = \pi_1(s_0)\lambda\gamma_1/(\mu\gamma_4)$$

$$\pi_1(s_5) = \pi_1(s_0)\lambda\gamma_1\nu q/(\mu\gamma_4)$$

$$\pi_1(s_6) = \pi_1(s_0)\lambda\gamma_1\nu(1-q)/(\mu\gamma_4)$$

It remains to derive $\pi_1(s_0)$ that is computed by normalising the probabilities. We can apply the same approach to derive the equilibrium distribution of P_2, obtaining:

$$\pi_2(t_1) = \pi_2(t_3) = \pi_2(t_0)\frac{1}{2}$$

$$\pi_2(t_2) = \pi_2(t_0)\frac{1}{2\alpha}$$

$$\pi_2(t_4) = \pi_2(t_0)\frac{1}{2\beta}$$

Again, by normalising the probabilities, we obtain $\pi_2(t_0)$. By applying Theorem 6.4 we can now easily derive the desired result:

$$\pi(s_5, t_2) + \pi(s_5, t_4) + \pi(s_6, t_2) + \pi(s_6, t_4) = \\ \pi_1(s_5)\pi_2(t_2) + \pi_1(s_5)\pi_2(t_4) + \pi_1(s_6)\pi_2(t_2) + \pi_1(s_6)\pi_2(t_4).$$

Notice that we have not built the joint state space and also that the automata P_1 and P_2 are not independent. For example, when P_2 is in state t_1 and P_1 is in checkpoint s_4, P_2 moves to t_2 only if P_1 decides neither to roll back to checkpoint s_1 nor to move to s_5.

6.5 Conclusion

In this chapter we have proposed an abstract modelling framework for the quantitative analysis of reversible computations. The main idea is to exploit the time-reversibility property of Markov processes in order to provide a computationally efficient way of deriving the desired performance indices. We have shown that, under some conditions, the proposed approach is suitable to be applied for a compositional formalism based on labelled stochastic automata. As a consequence the advantages (reduction of time-complexity and improvement of algorithms' numerical stability) of time-reversibility are applicable also for the analysis of the cooperation of automata that are proved to have product-form steady-state distributions.

Chapter 7

Product-form for models at discrete time

7.1 Introduction

Probabilistic I/O automata (PIOAs) provide a modelling framework that is well suited for describing and analysing distributed and concurrent systems. They incorporate a notion of probabilistic choice as well as a notion of composition that allows one to construct a PIOA for a composite system from a collection of simpler PIOAs representing the components. Differently from other probabilistic models, the local actions of a PIOA are associated with time delays governed by independent random variables with continuous-time exponential distributions. Our contribution consists in studying the product-form property for PIOAs. Our main result is the formulation of a theorem giving sufficient conditions for a composition of PIOAs to be in product-form and hence to efficiently compute its stationary probabilities.

7.2 Motivations

Probabilistic Input/Output automata (PIOAs) have been introduced in [102, 101] as a formalism aimed at modelling distributed and concurrent systems in a compositional way. However, the interest for their application goes beyond the purely engineering applications [13]. PIOAs incorporate a notion of probabilistic choice and time delays for locally controlled actions that distinguish them from earlier work [82]. The definition of formalisms for modelling probabilistic systems has been extensively investigated in the literature both in the field of process algebras and automata theory. One of the key-factors that characterises the proposed methodologies is clearly the semantics of the composition. Giving a reasonable way of composing probabilistic systems is challenging because the probabilities that are specified within each single component have a "local" meaning. In general, they are not sufficient to describe the probabilistic behaviour of the joint model without further assumptions such as the time scale. In the PIOA model this problem is solved by associating an exponentially distributed delay parameter with each state. Intuitively, a PIOA first draws a random delay time from an independent exponentially distributed random variable and then performs the probabilistic choice. Therefore, in the composition of a collection of PIOAs, the usual *race condition policy* used in [70, 97] is applied. Another feature of PIOAs is the way they handle the synchronisations which

is inspired by the I/O automata originally defined in [82]. PIOAs communicate via input and output actions and can perform internal non-communicating transitions. The communication is seen as a message transmitted on a labelled channel (that we call synchronisation label) by the output automaton. The synchronising automaton can read the message and perform a probabilistic transition accordingly. For each PIOA the sum of the probabilities associated with output and internal transitions, called locally controlled transitions, must be 1. On the other hand, upon the reception of a message, the PIOA immediately reacts, i.e., the sum of the probabilities associated with the message-receiving transitions outgoing from each state of a PIOA must be 1 for each of them separately (including the possible self-loops).
PIOAs share with many other formalisms for the quantitative analysis of computer systems the property of having an underlying Markov process that describes the model evolution, and the problem of the exponential growth of the cardinality of the state spaces which makes the derivation of the quantitative indices unfeasible even for relatively small systems. The problem of defining compositional approaches to the quantitative analysis of PIOAs has been addressed in [102] for what concerns the transient behaviour. To the best of our knowledge, the problem of defining a compositional approach for studying the stationary behaviour of PIOAs remains open. In the literature of queueing networks this problem is often associated with the so called *product-form* analysis which is described in [76] and then extended in numerous subsequent works (see, e.g., [46, 43, 11]). In the last decade the product-form approach has been successfully extended to include Markovian process algebras [61, 66]. Informally, a product-form model can be studied without constructing the stochastic process underlying the composition of the simpler components forming the systems, but these can be studied in isolation. Hence, the computational effort required to compute the stationary quantitative indices is highly reduced. Our contribution consists in studying the product-form property for PIOAs. Our main result is the formulation of a theorem giving sufficient conditions for a composition of PIOAs to be in product-form and hence to efficiently compute the stationary probabilities.

7.2.1 Characterisation of Product-forms

The literature about the characterisation of product-forms for various formalisms is very rich. Since the pioneering work of Kelly [76], several other works have addressed the problem of characterising the product-form of queueing networks in terms of different properties. These works have been extended with the introduction of Gelenbe's G-Networks [46] whose characterisation of the product-form is surveyed in [27, 83]. Similar efforts have been devoted to stochastic Petri nets product-forms [10] and Markovian process algebra [61]. The common denominator among all these contributions is that the considered models are based on continuous-time transition rates. Fewer results are available for probabilistic models expressed in terms of probabilistic process algebras or probabilistic automata. In the latter context we mention the results by Fourneau in [39, 40] but the synchronisation semantics which is considered is different from that of PIOAs and hence the results are not directly applicable.

7.2.2 Embedded Marov Chains and Uniformization

In many cases it is possible to reduce the analysis of a CTMC to that of a corresponding DTMC. There are two main approaches to associate a DTMC with a CTMC (see, e.g., [104]):

- The uniformisation method;
- The construction of the so called embedded chain.

A CTMC is fully characterised by its infinitesimal generator matrix $\mathbf{Q}$ where q_{ij} is the transition rate from state i to j for $i \neq j$ while the diagonal elements $q_{ii} = -q_i$ are defined as the negative sum of the non-diagonal elements of each row while the DTMC is fully characterised by its transition probability matrix $\mathbf{P}$ where p_{ij} is the transition probability from state i to j for $i \neq j$ while the diagonal elements p_{ii} is the probability to remain in the state i.
Let $X(t)$ be a CTMC such that there exists a positive real number ϕ as an upper limit for its rates (i.e. $q_i \leq \phi$ for all $i \in S$). We define the DTMC $X^U(t)$ with the same state space of $X(t)$ by uniformisation, i.e., we consider the maximum rate ν that appears in the CT system (i.e. $\nu = \max\{q_i, i \in S\}$) and define the transition probabilities of $X^U(t)$ as follows:

$$p_{ij} = \begin{cases} \frac{q_{ij}}{\nu} & \text{if } i \neq j \\ 1 - \frac{q_i}{\nu} & \text{if } i = j \,. \end{cases}$$

Thus the steady-state probability distribution of $X^U(t)$ and $X(t)$ are the same and sometimes it may be more convenient to study the uniformised chain rather then that at continuous time.

Another possible way of analysing a CTMC $X(t)$ is through the corresponding *embedded* Markov chain $X^E(t)$. If we consider the Markov process only at the moments upon which the state of the system changes, and we number these instances 0, 1, 2, etc., then we get a DTMC. This Markov chain has the transition probabilities p_{ij} for $i, j \in S$ as:

$$p_{ij} = \frac{q_{ij}}{\sum_{k \neq i} q_{ik}} \text{ for } j \neq i \tag{7.1}$$

and $p_{ii} = 0$. If $\boldsymbol{\pi}$ is the steady-state distribution of the DTMC (notice that it maybe periodic) one may derive the distribution $\boldsymbol{\pi}^*$ (which coincides in the continuous time with the limiting distribution even if the embedded DTMC is periodic) of the corresponding CTMC, assuming its ergodicity, as:

$$\pi_i^* = \frac{\pi_i q_i^{-1}}{\sum_{i \in S} \pi_i q_i^{-1}} \,. \tag{7.2}$$

With the uniformisation we must know the entire system to compute the uniformisation of a single state but with the corresponding embedded chain we only have to know the rates of a single state. Both of them unlickily aren't bijective functions, e.g. a system will have the same uniformized or embedded chain with respect to a system with the same state space and with doubled rates. As we can see in Figure 7.1 and in Figure 7.2. This behaviour can be explained as the systems, in the continuous case, have different rates but the transitions take place with the same proportion. This difference is flattened passing in the discrete case in which transitions may occur only in uniform slots of time.
We will see that the definition of the embedded DTMC has an important role in the interpretation of the synchronisation among PIOAs.

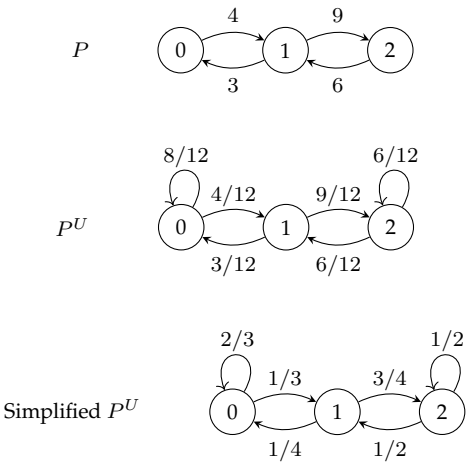
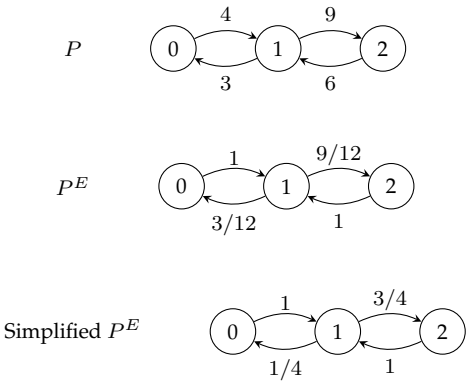


Figure 7.1: Two examples of CTMC Uniformization.



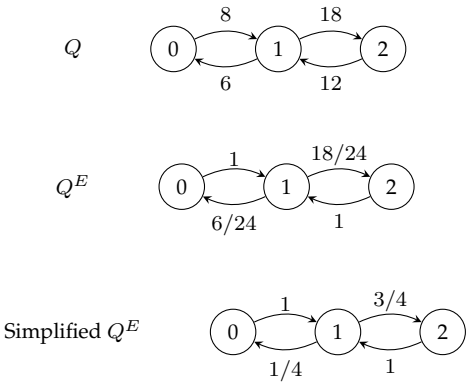


Figure 7.2: Two examples of CTMC Embedded chains.

7.2.3 Stochastic Automata

As we saw in Section 3.3, many high-level specification languages for stochastic discrete-event systems are based on labelled automata [16, 69, 70, 97] equipped with a composition operator and timed actions whose delays are governed by independent random variables with continuous-time exponential distributions. Models of stochastic automata presented in [88, 89] which draw a distinction between *active* and *passive* action types, and forming the composition of automata, only active/passive synchronisations are permitted. An analogous semantics is proposed for Stochastic Automata Networks (SAN) in [97].

7.2.4 PIOAs

As we saw in Section 3.4, in [102, 101], Smolka, Stark and Wu defined the class of *probabilistic I/O automata* (PIOA) which are a model for distributed or concurrent systems that incorporates a notion of probabilistic choice together with a composition rule. This model is based on a combination of reactive and generative transitions. In a reactive system, probabilities are distributed over the outgoing transitions labelled with the same action, i.e., actions are treated as being provided by the environment and there are no probabilistic assumptions about the behaviour of the environment. On the other hand, in a generative system, probabilities are distributed over all outgoing transitions from a state, i.e., actions are treated as locally generated by the system. Note that in a generative system there is no non-determinism present, while in a reactive system there is only external non-determinism.
In a probabilistic I/O automaton for every input action there is a reactive transition. Moreover, it is assumed that each input action is enabled in each state of a PIOA. The output and internal actions (called locally controlled actions) are treated generatively. At most one generative probabilistic transition gives the local behaviour of each state. A delay rate parameter δ is also added to each state.
PIOAs are an extension of the I/O model of Lynch and Tuttle [82] with probabilities. Our simplified version of the definition is equivalent to that in [101] in the case of "ergodic" PIOAs, which are those we considered in our work.

7.2.4.1 Delay Function

The state delay function δ_P is explained as follows: upon arrival in a state s, the PIOA P chooses randomly the length of time it will spend in that state before executing its next locally controlled (internal or output) transition. The random choice is made, independently of the other PIOAs in the system, according to an exponential holding time distribution whose mean is the reciprocal $1/\delta_P(s)$ of the delay parameter $\delta_P(s)$ associated with that state. If no locally controlled actions are enabled in this state then $\delta_P(s) = 0$.

7.3 Discretization of a SA into a PIOA

In this section we study the relations between SAs and PIOAs. First we present a method to transform a stochastic automaton P into a probabilistic one P^D. The *Discretization* is a method we introduce to transform a Stochastic Automata into a Probabilistic Input/Output Automata. This method relies on the embedded chain method for what

concerns the transformation between transition rates and transition probabilities. Informally, we consider the stochastic automata only at the moments at which the state of the system changes, and we number these instances 0, 1, 2, etc., then we get a probabilistic automata. Moreover we add the delay rate at each state of the probabilistic automata (representing the waiting time between each change of the system) and we obtain a probabilistic Input/Output automata.

Each state of P^D is equipped with a *delay rate* representing the waiting time between each change of the system. We show that discretization is indeed a bijection from the class of SAs to the class of PIOAs. Then, we prove that, under the assumption that all input actions synchronise with output ones in the PIOA model, discretization respects the synchronisation in the sense that, given two stochastic automata, the composition of the corresponding discretized automata coincides with the discretization of the composition of the two stochastic automata.

Definition 7.1. *(Discretization of a SA into a PIOA)* Given a stochastic automaton $P = (\mathcal{S}_P, \mathcal{A}ct_P, Pass_P, \rightsquigarrow_P, q_P)$, the discretization of P is the probabilistic I/O automaton $P^D = (\mathcal{S}_{P^D}, Act_{P^D}, Pass_{P^D}, \rightsquigarrow_{P^D}, \mu_{P^D}, \delta_{P^D})$ defined as follows:

- $\mathcal{S}_{P^D} = \mathcal{S}_P$
- $\mathcal{A}ct_{P^D} = \mathcal{A}ct_P$,
- $\mathcal{P}ass_{P^D} = \mathcal{P}ass_P$
- $\{\tau\}_{P^D} = \{\tau\}_P$
- $\rightsquigarrow_{P^D} = \rightsquigarrow_P$
- $\delta_{P^D}(s) = \sum_{a \in \mathcal{A}_P} q_P(s, a)$ with $s \in \mathcal{S}_P$
- μ_{P^D} is the transition probability function from $\rightsquigarrow_{P^D}$ to $(0, 1]$ such that
  - for all $s_1, s_2 \in \mathcal{S}_P$ and for all $a \in \mathcal{P}ass_P = \mathcal{P}ass_{P^D}$, $\mu_{P^D}(s_1, s_2, a) = q_P(s_1, s_2, a)$ (since $q_P(s_1, s_2, a)$ is already a probability)
  - for all $s_1, s_2 \in \mathcal{S}_P$ and for all $a \in \mathcal{L}_P = \mathcal{L}_{P^D}$, $\mu_{P^D}(s_1, s_2, a) = q_P(s_1, s_2, a)/\delta_{P^D}(s_1)$.

Informally, for each transition function $i \xrightarrow{(a,q)}_P j$ with $i, j \in \mathcal{S}_P$, this automata has the following transition probabilities p_{ij}:

- if $a \in \mathcal{A}ct_P$ then $p_{ij} = \dfrac{q_{ij}}{\delta_{P^C}(i)}$
- if $a \in \mathcal{P}ass_P$ then $p_{ij} = q_{ij}$ since q_{ij} is already a probability (e.g. $q_{ij} = \top$ corresponds to $p_{ij} = 1$

We can define q_P in terms of the two functions $q_{\mathcal{P}ass_P}$ and $q_{\mathcal{A}ct_P}$, such that:

$$\forall s_1, s_2 \in \mathcal{S}_P \text{ we have } q_P = \begin{cases} q_{\mathcal{P}ass_P} = [q_P(s_1, s_2, a) | a \in \mathcal{P}ass_P] \\ q_{\mathcal{A}ct_P} = [q_P(s_1, s_2, a) | a \in \mathcal{A}ct_P] \end{cases}$$

In this way we have:

- $q_{\mathcal{P}ass_{P^C}} = q_{\mathcal{P}ass_P}$

- $q_{\mathcal{A}ct_{PC}} = \dfrac{q_{\mathcal{A}ct_P}}{\delta_{PC}}$

$\dfrac{q_{\mathcal{A}ct_P}}{\delta_{PC}}$ is the concise notation of $\dfrac{q_{s,s'}}{\delta_{PC}(s)}$ for all $s \in \mathcal{S}_P$ and for all $q_{s,s'} \in q_{\mathcal{A}ct_P}$.

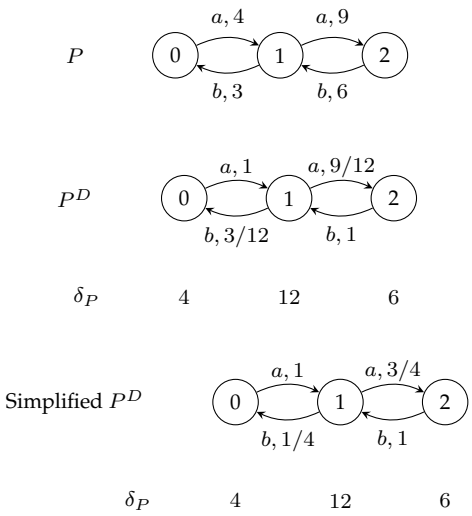
Notice that $\delta_{PD}(s)$ is the sum of the transition rates of the active actions of P outgoing from s. If s has only passive actions outgoing from it then $\delta_{PD}(s) = 0$. In this way we don't have to know the entire system to compute the discretization of a single state but only the rates of a single state. Moreover, if P is closed then also P^D is.
Differently from the uniformization method and the embedded chain, discretization is a bijective function from the set of all SAs to the set of all PIOAs. The inverse of the discretization function allows one to transform a probabilistic I/O automaton into the unique stochastic automaton having it as the corresponding discretization. Two systems with the same state space and with doubled rates will have the same uniformized or embedded chain however in case of the discretization, they will have the same probabilities but different δ functions, as we can see in Figure 7.3. Moreover, when the transition rates are doubled also the corresponding δs double too. This behaviour can be explained as the uniformization and the embedded chain methods are functions from a class of CTMC (with different transition rates but the same proportions) to a single DTMC. On the contrary, discretization method keeps track of which chain of the CTMC class it derives from. As we will see this property is also kept from DTMC to CTMC.

Proposition 10. (Bijection of Discretization) The discretization transformation of Definition 7.1 is a bijection from the set of all SAs to the set of all PIOAs.

Proof. It is easy to prove that discretization is an injective function, i.e., if P_1 and P_2 are two distinct stochastic automata then $P_1^D \neq P_2^D$. Indeed, assume by contradiction that $P_1^D = P_2^D$. Then P_1 and P_2 have the same state space, the same sets of active and passive action types, the same transition relation and the same probabilities associated to passive type transitions. From the fact that P_1^D and P_2^D have the same delay function it also follows that P_1 and P_2 have the same rates associated to locally controlled transitions. This contradicts the hypothesis that P_1 and P_2 are distinct. (Informally, the discretization method consists in equalities, a summation and a division between real numbers. All of these operators have trivially the injection property therefore also the discretization has the injection property.)
In order to prove that discretization is a surjective function consider the following transformation from PIOAs to SAs. Let $P = (\mathcal{S}_P, Act_P, Pass_P, \rightsquigarrow_P, \mu_P, \delta_P)$ be a PIOA. Define the stochastic automaton $P^S = (\mathcal{S}_{PS}, Act_{PS}, Pass_{PS}, \rightsquigarrow_{PS}, q_{PS})$ defined as follows:

- $\mathcal{S}_{PS} = \mathcal{S}_P$
- $\mathcal{A}ct_{PS} = \mathcal{A}ct_P$
- $\mathcal{P}ass_{PS} = \mathcal{P}ass_P$
- $\rightsquigarrow_{PS} = \rightsquigarrow_P$
- q_{PS} is the function from $\rightsquigarrow_{PD}$ to $\mathbb{R}^+$ such that
  - for all $s_1, s_2 \in \mathcal{S}_P$ and for all $a \in \mathcal{P}ass_{PS} = \mathcal{P}ass_P$, $q_{PS}(s_1, s_2, a) = \mu_P(s_1, s_2, a)$
  - for all $s_1, s_2 \in \mathcal{S}_P$ and for all $a \in \mathcal{L}_{PS} = \mathcal{L}_P$, $q_{PS}(s_1, s_2, a) = \mu_P(s_1, s_2, a)\delta_P(s_1)$.

It is easy to prove that for every probabilistic input/output automaton P, the discretization of P^S coincides with P. □



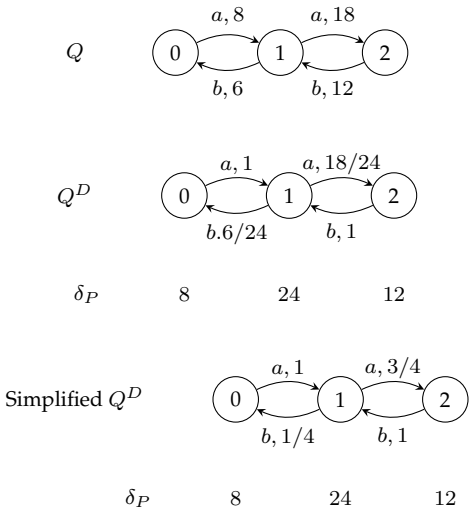


Figure 7.3: Two examples of CTMC discretization with:
$\mathcal{A}ct_P = \mathcal{A}ct_Q = \{a, b\}$
$\mathcal{P}ass_P = \mathcal{P}ass_Q = \emptyset$.

The following corollary will be useful to prove the next theorem.

Corollary 3. Let P be a closed SA and P^D be the corresponding discretized PIOA (defined according to Definition 7.1). Then

- for all $s \in \mathcal{S}_P$, $\delta_{P^D}(s) = q_P(s)$
- for all $s \in \mathcal{S}_P$ and for all $a \in (Act_P \cup Pass_P)$, $q_P(s, s', a) = \mu_P(s, s', a)\, \delta_{P^D}(s)$.

The stationary distribution of a closed SA P can be derived from that of the corresponding discretized automaton P^D and vice versa.

Theorem 7.2. *(Relations between stationary distributions of SA and PIOA) Let P be a closed SA and P^D be the corresponding discretized PIOA (defined according to Definition 7.1). Let $\mathcal{S} = \mathcal{S}_P = \mathcal{S}_{P^D}$.*

- *If $\boldsymbol{\pi}_{P^D}$ is the stationary distribution of the DTMC underlying P^D then $\boldsymbol{\pi}_P$ defined by*

$$\pi_P(s) = \frac{\pi_{P^D}(s)\delta^{-1}_{P^D}(s)}{\sum_{s\in\mathcal{S}} \pi_{P^D}(s)\delta^{-1}_{P^D}(s)} \tag{7.3}$$

  for all $s \in \mathcal{S}$, is the stationary distribution of the CTMC underlying P.

- *If $\boldsymbol{\pi}_P$ is the stationary distribution of the CTMC underlying P then $\boldsymbol{\pi}_{P^D}$ defined by*

$$\pi_{P^D}(s) = \frac{\pi_P(s)\delta_{P^D}(s)}{\sum_{s\in\mathcal{S}} \pi_P(s)\delta_{P^D}(s)} \tag{7.4}$$

  for all $s \in \mathcal{S}$, is the stationary distribution of the DTMC underlying P^D.

Proof. In order to prove the first statement, consider the GBEs of the CTMC underlying P:

$$\pi_P(s) \sum_{s'\in\mathcal{S}_P, s'\neq s} q_P(s, s') = \sum_{s'\in\mathcal{S}_P, s'\neq s} \pi_P(s')\, q_P(s', s).$$

Adding the self-loops we get:

$$\pi_P(s) \sum_{s'\in\mathcal{S}_P} q_P(s, s') = \sum_{s'\in\mathcal{S}_P} \pi_P(s') q_P(s', s).$$

We can now replace $\pi_P(s)$ with its definition of Equation (7.3) obtaining:

$$\frac{\pi_{P^D}(s)\delta^{-1}_{P^D}(s)}{\sum_{s''\in\mathcal{S}_P} \pi_{P^D}(s'')\delta^{-1}_{P^D}(s'')} \sum_{s'\in\mathcal{S}_P} q_P(s, s') = \sum_{s'\in\mathcal{S}_P} \frac{\pi_{P^D}(s')\delta^{-1}_{P^D}(s')}{\sum_{s''\in\mathcal{S}_P} \pi_{P^D}(s'')\delta^{-1}_{P^D}(s'')} q_P(s', s).$$

From Corollary 3, since $\sum_{s'\in\mathcal{S}_P} q_P(s, s') = q_P(s) = \delta_{P^D}(s)$ and $q_P(s', s) = \mu_P(s', s)\, \delta_{P^D}(s')$ we have:

$$\frac{\pi_{P^D}(s)\delta^{-1}_{P^D}(s)}{\sum_{s''\in\mathcal{S}_P} \pi_{P^D}(s'')\delta^{-1}_{P^D}(s'')} \delta_{P^D}(s) = \sum_{s'\in\mathcal{S}_P} \frac{\pi_{P^D}(s')\delta^{-1}_{P^D}(s')}{\sum_{s''\in\mathcal{S}_P} \pi_{P^D}(s'')\delta^{-1}_{P^D}(s'')} \mu_P(s', s)\, \delta_{P^D}(s').$$

By simplyfing, we obtain:

$$\pi_{P^D}(s)\frac{1}{\sum_{s''\in\mathcal{S}_P}\pi_{P^D}(s'')\delta_{P^D}^{-1}(s'')} = \frac{1}{\sum_{s''\in\mathcal{S}_P}\pi_{P^D}(s'')\delta_{P^D}^{-1}(s'')}\sum_{s'\in\mathcal{S}_P}\pi_{P^D}(s')\,\mu_P(s',s)$$

that is

$$\pi_{P^D}(s) = \sum_{s'\in\mathcal{S}_P}\pi_{P^D}(s')\,\mu_P(s',s)$$

which are exactly the GBEs for the DTMC underlying P^D. Since, if it exists, the stationary distribution of P^D is unique, we can conclude that also the second statement of our proposition holds.
The proof of the second statement is analogous by using the GBEs of the DTMC underlying P^D, replacing $\pi_{P^D}(s)$ with its definition of Equation (7.4) and finding the GBEs of the CTMC underlying P. □

Another interesting property of the discretization function is that it respects the synchronisation operator when we assume that all the input actions synchronise with the output ones in the PIOA model.

Proposition 11 (Discretization respects the synchronisation)**.** Let P and Q be two SAs and P^D and Q^D be the corresponding discretized PIOAs. Assume that $\mathcal{P}ass_P = \mathcal{A}ct_Q$ and $\mathcal{P}ass_Q = \mathcal{A}ct_P$. Then

$$(P\otimes Q)^D = P^D\otimes Q^D.$$

Proof. It is easy to prove that $(P\otimes Q)^D$ and $P^D\otimes Q^D$ are both closed and have the same state space $\mathcal{S}_P\times\mathcal{S}_Q$, the same set of output actions $\mathcal{A}ct_P\cup\mathcal{A}ct_Q$ and the same transition relation. We prove that they have also the same delay functions and the same transition probabilities. By Definitions 3.3 and 7.1:

$$\delta_{(P\otimes Q)^D}((s_p,s_q)) = \sum_{a\in\mathcal{L}_{P\otimes Q}} q_{P\otimes Q}((s_p,s_q),a)$$

and

$$\delta_{P^D\otimes Q^D}((s_p,s_q)) = \delta_{P^D}(s_p)+\delta_{Q^D}(s_q) = \sum_{a\in\mathcal{A}_P} q_P(s_p,a)+\sum_{a\in\mathcal{A}_Q} q_Q(s_q,a)$$

From the semantics for pairwise SA synchronisations and the fact that for all $a\in\mathcal{P}ass_P$ (resp. $\mathcal{P}ass_Q$), $\sum_{s_2:(s_1,s_2,a)\in\rightsquigarrow_P} q_P(s_1,s_2,a) = 1$ (resp. $\sum_{s_2:(s_1,s_2,a)\in\rightsquigarrow_Q} q_P(s_1,s_2,a) = 1$) we can conlude that $\delta_{(P\otimes Q)^D}$ and $\delta_{P^D\otimes Q^D}$ coincide.

Now let:

- $a\in\mathcal{A}_P = \mathcal{P}_Q$;
- $(s_{p_1},s_{q_1}) \xrightarrow{(a,pr)}_{P\otimes Q} (s_{p_2},s_{q_2})$ because $s_{p_1}\xrightarrow{(a,r)}_P s_{p_2}$;
- $s_{q_1}\xrightarrow{(a,p)}_Q s_{q_2}$.

By Definition 7.1 we have that:

$$(s_{p_1},s_{q_1}) \xrightarrow{(a,pr/\delta_{(P\otimes Q)^D}(s_{p_1},s_{q_1}))}_{(P\otimes Q)^D} (s_{p_2},s_{q_2})$$

In this case,

$$s_{p_1} \xrightarrow{(a,r/\delta_{P^D}(s_{p_1}))}_{P^D} s_{p_2}$$

and

$$s_{q_1} \xrightarrow{(a,p)}_{Q^D} s_{q_2}$$

and then, by the semantics of PIOA synchronisation:

$$(s_{p_1}, s_{q_1}) \xrightarrow{(a,\Delta_1 pr/\delta_{P^D}(s_{p_1}))}_{P^D \otimes Q^D} (s_{p_2}, s_{q_2})$$

where $\Delta_1 = \delta_{P^D}(s_{p_1})/(\delta_{P^D}(s_{p_1}) + \delta_{Q^D}(s_{q_1}))$.
Hence, by simplifying, $(s_{p_1}, s_{q_1}) \xrightarrow{(a,pr/\delta_{P^D \otimes Q^D}(s_{p_1},s_{q_1}))}_{P^D \otimes Q^D} (s_{p_2}, s_{q_2})$.
The case $a \in \mathcal{P}_P = \mathcal{A}_Q$ is analogous. Consider now :

- $a = \tau$;
- $(s_{p_1}, s_{q_1}) \xrightarrow{(\tau,r)}_{P \otimes Q} (s_{p_2}, s_{q_2})$ because $s_{p_1} \xrightarrow{(\tau,r)}_{P} s_{p_2}$.

By Definition 7.1:

$$(s_{p_1}, s_{q_1}) \xrightarrow{(\tau,r/\delta_{(P\otimes Q)^D}(s_{p_1},s_{q_1}))}_{(P\otimes Q)^D} (s_{p_2}, s_{q_2})$$

In this case,

$$s_{p_1} \xrightarrow{(\tau,r/\delta_{P^D}(s_{p_1}))}_{P^D} s_{p_2}$$

and then, by the semantics of PIOA synchronisation,

$$(s_{p_1}, s_{q_1}) \xrightarrow{(a,\Delta_1 r/\delta_{P^D}(s_{p_1}))}_{P^D \otimes Q^D} (s_{p_2}, s_{q_2})$$

where $\Delta_1 = \delta_{P^D}(s_{p_1})/(\delta_{P^D}(s_{p_1}) + \delta_{Q^D}(s_{q_1}))$.
Hence, by simplifying, $(s_{p_1}, s_{q_1}) \xrightarrow{(a,r/\delta_{P^D \otimes Q^D}(s_{p_1},s_{q_1}))}_{P^D \otimes Q^D} (s_{p_2}, s_{q_2})$. □

Example 2. Consider the stochastic automata P and Q depicted in Figure 7.4 with $\mathcal{A}ct_P = \mathcal{P}ass_Q = \{a\}$ and $\mathcal{P}ass_P = \mathcal{A}ct_Q = \emptyset$. Let $P \otimes Q$ be the composition of P and Q defined according to Definition 3.3 and $(P \otimes Q)^D$ be the probabilistic I/O automaton corresponding to the discretization of $P \otimes Q$. The discretizations of P and Q are depicted in Figure 7.5 together with their probabilistic composition. One can verify that $(P \otimes Q)^D = P^D \otimes Q^D$.

7.4 Product-form for PIOA

In this section we present our product-form result for probabilistic I/O automata. In particular we prove that the stationary distribution of the composition of two PIOAs, P and Q, can be computed without constructing the stochastic process underlying the whole system $P \otimes Q$, but it can be derived from the the stationary distribution of the two components in isolation.
We first introduce a closure operation over probabilistic I/O automata that allows us to assign to all the transitions with the same input action a the same transition probability λ.

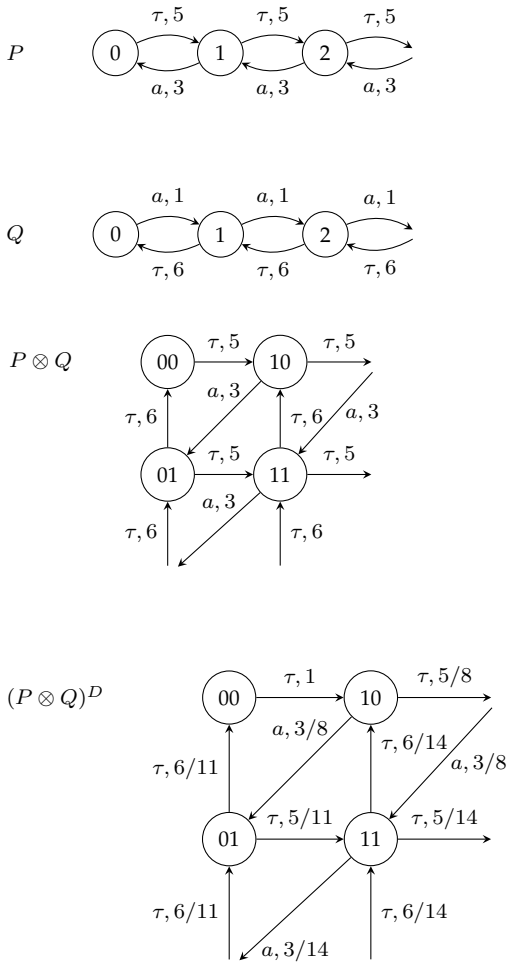
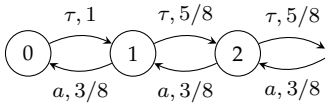


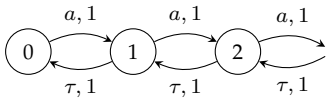
Figure 7.4: Example of $(P \otimes Q)^D$

P^D



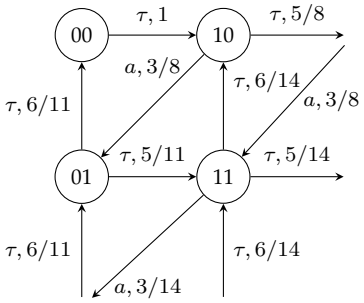
with $\delta(0) = 5$, $\delta(1) = 8$, $\delta(2) = 8$

Q^D



with $\delta(0) = 0$, $\delta(1) = 6$, $\delta(2) = 6$

$P^D \otimes Q^D$



with $\delta(00) = 5$, $\delta(10) = 8$,
$\delta(01) = 11$, $\delta(11) = 14$.

Figure 7.5: Example of $P^D \otimes Q^D$

Definition 7.3. *(PIOA closure)* The closure of a probabilistic I/O automaton P with respect to an input action $a \in \mathcal{P}ass_P$ and $\lambda \in \mathbb{R}^+$, written $P^C = P\{a \leftarrow \lambda\}$, is the PIOA defined as follows:

- $\mathcal{S}_{P^C} = \mathcal{S}_P$
- $\mathcal{A}ct_{P^C} = \mathcal{A}ct_P \cup \{a\}$
- $\mathcal{P}ass_{P^C} = \mathcal{P}ass_P \setminus \{a\}$
- $\rightsquigarrow_{P^C} = \rightsquigarrow_P$
- μ_{P^C} is the transition probability function from $\rightsquigarrow_{P^C}$ to $(0, 1]$ such that
  - for all $s_1, s_2 \in \mathcal{S}_P$ such that $(s_1, s_2, a) \in \rightsquigarrow_P$

$$\mu_{P^C}(s_1, s_2, a) = \mu_P(s_1, s_2, a)\frac{\lambda}{\delta_P(s_1) + \lambda}$$

  - for all $s_1, s_2 \in \mathcal{S}_P$ and for all $b \in \mathcal{P}ass_P \setminus \{a\}$,

$$\mu_{P^C}(s_1, s_2, b) = \mu_P(s_1, s_2, b)$$

  - for all $s_1, s_2 \in \mathcal{S}_P$ and for all $b \in \mathcal{L}_P$,

$$\mu_{P^C}(s_1, s_2, b) = \mu_P(s_1, s_2, b)\frac{\delta_P(s_1)}{\delta_P(s_1) + \lambda}$$

- δ_{P^C} is such that for all $s \in \mathcal{S}_P$, $\delta_{P^C}(s) = \delta_P(s) + \lambda$.

Several closures can be specified by susequently applying Definition 7.3. It is easy to prove that the order in which the closures are applied is irrelevant and then, if $\mathcal{P}ass_P = \{a_1, \ldots, a_n\}$ and $\{\lambda_1, \ldots, \lambda_n\}$ is a set of positive real numbers then we write $P\{a_i \leftarrow \lambda_i\}_{a_i \in \mathcal{I}_P}$ for $((P\{a_1 \leftarrow \lambda_1\}) \cdots)\{a_n \leftarrow \lambda_n\}$.

We are now ready to prove the product-form theorem.

Theorem 7.4. (Product-forms for PIOA) *Let P and Q be two probabilistic I/O automata such that $\mathcal{A}ct_P = \mathcal{P}ass_Q$ and $\mathcal{A}ct_Q = \mathcal{P}ass_P$. Let $\{a_1, \ldots, a_n\} = \mathcal{A}ct_P \cup \mathcal{A}ct_Q$. If there exists a set of positive real numbers $\{\lambda_1, \ldots, \lambda_n\}$ such that $P^C = P\{a_i \leftarrow \lambda_i\}_{a_i \in \mathcal{P}ass_P}$ and $Q^C = Q\{a_i \leftarrow \lambda_i\}_{a_i \in \mathcal{P}ass_Q}$ satisfy the following equations:*

- *for all $s_p \in \mathcal{S}_P$ and for all $a_i \in \mathcal{A}ct_P$,*

$$\delta_{P^C}(s_p) \sum_{s'_p \in \mathcal{S}_{P^C}} \mu_{P^C}(s'_p, s_p, a_i)\frac{\pi_{P^C}(s'_p)}{\pi_{P^C}(s_p)}\frac{\delta_P(s'_p)}{\delta_{P^C}(s'_p)} = \lambda_i \tag{7.5}$$

- *for all $s_q \in \mathcal{S}_Q$ and for all $a_i \in \mathcal{A}ct_Q$,*

$$\delta_{Q^C}(s_q) \sum_{s' \in \mathcal{S}_{Q^C}} \mu_{Q^C}(s'_q, s_q, a_i)\frac{\pi_{Q^C}(s'_q)}{\pi_{Q^C}(s_q)}\frac{\delta_Q(s'_q)}{\delta_{Q^C}(s'_q)} = \lambda_i \tag{7.6}$$

then for all $(s_p, s_q) \in \mathcal{S}_{P\otimes Q}$

$$\pi_{P\otimes Q}(s_p, s_q) \propto \pi_{P^C}(s_p)\pi_{Q^C}(s_q)\frac{\delta_P(s_p)+\delta_Q(s_q)}{\delta_{P^C}(s_p)+\delta_{Q^C}(s_q)}\,. \tag{7.7}$$

Proof. For the sake of readability, we assume that $\mathcal{A}ct_P = \mathcal{P}ass_Q = \{a\}$ and $\mathcal{A}ct_Q = \mathcal{P}ass_P = \{b\}$ and that for all $s_p \in \mathcal{S}_P$ there exists a unique $s'_p \in \mathcal{S}_P$ such that $(s_p, s'_p, b) \in \leadsto_P$, i.e., $\mu_P(s_p, s'_p, b) = 1$, and for all $s_q \in \mathcal{S}_Q$ there exists a unique $s'_q \in \mathcal{S}_Q$ such that $(s_q, s'_q, a) \in \leadsto_P$, i.e., $\mu_Q(s_q, s'_q, a) = 1$. The proof can be easily generalized to the case of an arbtrary number of output actions in P and in Q, respectively. Let $P^C = P\{b \leftarrow \lambda_b\}$ and $Q^C = Q\{a \leftarrow \lambda_a\}$ where λ_a and λ_b are defined according to Equations (7.5) and (7.6). The global balance equations (GBEs) for a state $s \in \mathcal{S}_{P^C}$ are:

$$\pi_{P^C}(s) = \sum_{s'\in\mathcal{S}_{P^C}} \mu_P(s', s, \tau)\frac{\delta_P(s')}{\delta_{P^C}(s')}\pi_{P^C}(s') + \sum_{s'\in\mathcal{S}_{P^C}} \mu_P(s', s, a)\frac{\delta_P(s')}{\delta_{P^C}(s')}\pi_{P^C}(s') + \\ + \sum_{s'\in\mathcal{S}_{P^C}} \frac{\lambda_b}{\delta_{P^C}(s')}\pi_{P^C}(s')$$

analogously, the GBEs for a state $s \in \mathcal{S}_{Q^C}$ are:

$$\pi_{Q^C}(s) = \sum_{s'\in\mathcal{S}_{Q^C}} \mu_Q(s', s, \tau)\frac{\delta_Q(s')}{\delta_{Q^C}(s')}\pi_{Q^C}(s') + \sum_{s'\in\mathcal{S}_{Q^C}} \mu_Q(s', s, b)\frac{\delta_Q(s')}{\delta_{Q^C}(s')}\pi_{q^C}(s') + \\ + \sum_{s'\in\mathcal{S}_{Q^C}} \frac{\lambda_a}{\delta_{Q^C}(s')}\pi_{Q^C}(s')$$

The GBEs for a joint state $(s_p, s_q) \in \mathcal{S}_{P\otimes Q}$ are

$$\begin{aligned}\pi_{P\otimes Q}(s_p, s_q) = &\sum_{s'_p\in\mathcal{S}_P} \mu_{P\otimes Q}((s'_p, s_q), (s_p, s_q), \tau)\pi_{P\otimes Q}(s'_p, s_q) + \\ &+ \sum_{s'_q\in\mathcal{S}_Q} \mu_{P\otimes Q}((s_p, s'_q), (s_p, s_q), \tau)\pi_{P\otimes Q}(s_p, s'_q) + \\ &+ \sum_{(s'_p,s'_q)\in\mathcal{S}_{P\otimes Q}} \mu_{P\otimes Q}((s'_p, s'_q), (s_p, s_q), a)\pi_{P\otimes Q}(s'_p, s'_q) + \\ &+ \sum_{(s'_p,s'_q)\in\mathcal{S}_{P\otimes Q}} \mu_{P\otimes Q}((s'_p, s'_q), (s_p, s_q), b)\pi_{P\otimes Q}(s'_p, s'_q)\,.\end{aligned}$$

Let us now introduce the following notation: for all $(s_p, s_q) \in \mathcal{S}_{P\otimes Q}$, we write

- $\Delta_{s_p,s_q} = \frac{\delta_P(s_p)}{\delta_P(s_p)+\delta_Q(s_q)}$
- $\Delta_{s_q,s_p} = \frac{\delta_Q(s_q)}{\delta_P(s_p)+\delta_Q(s_q)}$.

By Definition 3.8 of PIOA synchronisation, we can write

$$\begin{aligned}\pi_{P\otimes Q}(s_p,s_q) = &\sum_{s'_p\in\mathcal{S}_P}\mu_P(s'_p,s_p,\tau)\Delta_{s'_p,s_q}\,\pi_{P\otimes Q}(s'_p,s_q)+\\ &+\sum_{s'_q\in\mathcal{S}_Q}\mu_Q(s'_q,s_q,\tau)\Delta_{s'_q,s_p}\,\pi_{P\otimes Q}(s_p,s'_q)+\\ &+\sum_{(s'_p,s'_q)\in\mathcal{S}_{P\otimes Q}}\mu_P(s'_p,s_p,a)\Delta_{s'_p,s'_q}\,\pi_{P\otimes Q}(s'_p,s'_q)+\\ &+\sum_{(s'_p,s'_q)\in\mathcal{S}_{P\otimes Q}}\mu_Q(s'_q,s_q,b)\Delta_{s'_q,s'_p}\,\pi_{P\otimes Q}(s'_p,s'_q)\end{aligned}$$

and by replacing $\pi_{P\otimes Q}(s_p,s_q)$ by the product-form of Equation (7.7)

$$\pi_{P^C}(s_p)\pi_{Q^C}(s_q)\frac{\delta_P(s_p)+\delta_Q(s_q)}{\delta_{P^C}(s_p)+\delta_{Q^C}(s_q)}$$

we obtain:

$$\begin{aligned}\delta_P(s_p)+\delta_Q(s_q) = \delta_{P^C}(s_p)&\sum_{s'_p\in\mathcal{S}_P}\mu_P(s'_p,s_p,\tau)\frac{\delta_P(s'_p)}{\delta_{P^C}(s'_p)}\frac{\pi_{P^C}(s'_p)}{\pi_{P^C}(s_p)}+\\ &+\delta_{Q^C}(s_q)\sum_{s'_q\in\mathcal{S}_Q}\mu_Q(s'_q,s_q,\tau)\frac{\delta_Q(s'_q)}{\delta_{Q^C}(s'_q)}\frac{\pi_{Q^C}(s'_q)}{\pi_{Q^C}(s_q)}+\\ &+\delta_{Q^C}(s_q)\sum_{s'_q\in\mathcal{S}_Q}\frac{1}{\delta_{Q^C}(s'_q)}\frac{\pi_{Q^C}(s'_q)}{\pi_{Q^C}(s_q)}\cdot\delta_{P^C}(s_p)\sum_{s'_p\in\mathcal{S}_P}\mu_P(s'_p,s_p,a)\frac{\delta_P(s'_p)}{\delta_{P^C}(s'_p)}\frac{\pi_{P^C}(s'_p)}{\pi_{P^C}(s_p)}+\\ &+\delta_{P^C}(s_p)\sum_{s'_p\in\mathcal{S}_P}\frac{1}{\delta_{P^C}(s'_p)}\frac{\pi_{P^C}(s'_p)}{\pi_{P^C}(s_p)}\cdot\delta_{Q^C}(s_q)\sum_{s'_q\in\mathcal{S}_Q}\mu_Q(s'_q,s_q,b)\frac{\delta_Q(s'_q)}{\delta_{Q^C}(s'_q)}\frac{\pi_{q^C}(s'_q)}{\pi_{Q^C}(s_q)}\end{aligned}$$

We now multiply the definitions of $\pi_{P^C}(s_p)$ and $\pi_{Q^C}(s_q)$ above by the factors $\delta_{P^C}(s_p)$ and $\delta_{Q^C}(s_q)$, respectively, and after few algebraic manipulations we get:

$$\begin{aligned}\delta_P(s_p) = \delta_{P^C}(s_p)&\sum_{s'_p\in\mathcal{S}_{P^C}}\mu_P(s'_p,s_p,\tau)\frac{\delta_P(s'_p)}{\delta_{P^C}(s'_p)}\frac{\pi_{P^C}(s'_p)}{\pi_{P^C}(s_p)}+\\ &+\lambda_a+\delta_{P^C}(s_p)\sum_{s'_p\in\mathcal{S}_{P^C}}\frac{\lambda_b}{\delta_{P^C}(s'_p)}\frac{\pi_{P^C}(s'_p)}{\pi_{P^C}(s_p)}-\lambda_b\end{aligned}$$

and

$$\begin{aligned}\delta_Q(s_q) = \delta_{Q^C}(s_q)&\sum_{s'_q\in\mathcal{S}_{Q^C}}\mu_Q(s'_q,s_q,\tau)\frac{\delta_Q(s'_q)}{\delta_{q^C}(s'_q)}\frac{\pi_{Q^C}(s'_q)}{\pi_{Q^C}(s_q)}+\\ &+\lambda_b+\delta_{Q^C}(s_q)\sum_{s'_q\in\mathcal{S}_{Q^C}}\frac{\lambda_a}{\delta_{Q^C}(s'_q)}\frac{\pi_{Q^C}(s'_q)}{\pi_{Q^C}(s_q)}-\lambda_a\end{aligned}$$

We replace the above definitions in the GBEs for $P \otimes Q$ and, after some simple algebraic operation, considering that

- $\delta_{P^C}(s_p) = \delta_P(s_p) + \lambda_b$
- $\delta_{Q^C}(s_q) = \delta_Q(s_q) + \lambda_a$

we derive:

$$\delta_{P^C}(s_p) \sum_{s'_p \in \mathcal{S}_{P^C}} \mu_P(s'_p, s_p, \tau) \frac{\delta_P(s'_p)}{\delta_{P^C}(s'_p)} \frac{\pi_{P^C}(s'_p)}{\pi_{P^C}(s_p)} + \lambda_a + \delta_{P^C}(s_p) \sum_{s'_p \in \mathcal{S}_{P^C}} \frac{\lambda_b}{\delta_{P^C}(s'_p)} \frac{\pi_{P^C}(s'_p)}{\pi_{P^C}(s_p)} - \lambda_b$$

$$+\delta_{Q^C}(s_q) \sum_{s'_q \in \mathcal{S}_{Q^C}} \mu_Q(s'_q, s_q, \tau) \frac{\delta_Q(s'_q)}{\delta_{q^C}(s'_q)} \frac{\pi_{Q^C}(s'_q)}{\pi_{Q^C}(s_q)} + \lambda_b + \delta_{Q^C}(s_q) \sum_{s'_q \in \mathcal{S}_{Q^C}} \frac{\lambda_a}{\delta_{Q^C}(s'_q)} \frac{\pi_{Q^C}(s'_q)}{\pi_{Q^C}(s_q)} - \lambda_a =$$

$$\delta_{P^C}(s_p) \sum_{s'_p \in \mathcal{S}_P} \mu_P(s'_p, s_p, \tau) \frac{\delta_P(s'_p)}{\delta_{P^C}(s'_p)} \frac{\pi_{P^C}(s'_p)}{\pi_{P^C}(s_p)} + \delta_{Q^C}(s_q) \sum_{s'_q \in \mathcal{S}_Q} \mu_Q(s'_q, s_q, \tau) \frac{\delta_Q(s'_q)}{\delta_{Q^C}(s'_q)} \frac{\pi_{Q^C}(s'_q)}{\pi_{Q^C}(s_q)}$$

$$+ \delta_{Q^C}(s_q) \sum_{s'_q \in \mathcal{S}_Q} \frac{1}{\delta_{Q^C}(s'_q)} \frac{\pi_{Q^C}(s'_q)}{\pi_{Q^C}(s_q)} \cdot \delta_{P^C}(s_p) \sum_{s'_p \in \mathcal{S}_P} \mu_P(s'_p, s_p, a) \frac{\delta_P(s'_p)}{\delta_{P^C}(s'_p)} \frac{\pi_{P^C}(s'_p)}{\pi_{P^C}(s_p)}$$

$$+ \delta_{P^C}(s_p) \sum_{s'_p \in \mathcal{S}_P} \frac{1}{\delta_{P^C}(s'_p)} \frac{\pi_{P^C}(s'_p)}{\pi_{P^C}(s_p)} \cdot \delta_{Q^C}(s_q) \sum_{s'_q \in \mathcal{S}_Q} \mu_Q(s'_q, s_q, b) \frac{\delta_Q(s'_q)}{\delta_{Q^C}(s'_q)} \frac{\pi_{q^C}(s'_q)}{\pi_{Q^C}(s_q)} .$$

We now simplify and obtain:

$$\delta_{P^C}(s_p) \sum_{s'_p \in \mathcal{S}_{P^C}} \frac{\lambda_b}{\delta_{P^C}(s'_p)} \frac{\pi_{P^C}(s'_p)}{\pi_{P^C}(s_p)} + \delta_{Q^C}(s_q) \sum_{s'_q \in \mathcal{S}_{Q^C}} \frac{\lambda_a}{\delta_{Q^C}(s'_q)} \frac{\pi_{Q^C}(s'_q)}{\pi_{Q^C}(s_q)} =$$

$$\delta_{Q^C}(s_q) \sum_{s'_q \in \mathcal{S}_Q} \frac{1}{\delta_{Q^C}(s'_q)} \frac{\pi_{Q^C}(s'_q)}{\pi_{Q^C}(s_q)} \cdot \delta_{P^C}(s_p) \sum_{s'_p \in \mathcal{S}_P} \mu_P(s'_p, s_p, a) \frac{\delta_P(s'_p)}{\delta_{P^C}(s'_p)} \frac{\pi_{P^C}(s'_p)}{\pi_{P^C}(s_p)} +$$

$$+ \delta_{P^C}(s_p) \sum_{s'_p \in \mathcal{S}_P} \frac{1}{\delta_{P^C}(s'_p)} \frac{\pi_{P^C}(s'_p)}{\pi_{P^C}(s_p)} \cdot \delta_{Q^C}(s_q) \sum_{s'_q \in \mathcal{S}_Q} \mu_Q(s'_q, s_q, b) \frac{\delta_Q(s'_q)}{\delta_{Q^C}(s'_q)} \frac{\pi_{q^C}(s'_q)}{\pi_{Q^C}(s_q)} .$$

By substituting the definitions of λ_a and λ_b given by Equations (7.5) and (7.6), we obtain

$$\delta_{P^C}(s_p) \sum_{s'_p \in \mathcal{S}_{P^C}} \frac{\lambda_b}{\delta_{P^C}(s'_p)} \frac{\pi_{P^C}(s'_p)}{\pi_{P^C}(s_p)} + \delta_{Q^C}(s_q) \sum_{s'_q \in \mathcal{S}_{Q^C}} \frac{\lambda_a}{\delta_{Q^C}(s'_q)} \frac{\pi_{Q^C}(s'_q)}{\pi_{Q^C}(s_q)} =$$

$$\delta_{Q^C}(s_q) \sum_{s'_q \in \mathcal{S}_Q} \frac{1}{\delta_{Q^C}(s'_q)} \frac{\pi_{Q^C}(s'_q)}{\pi_{Q^C}(s_q)} \lambda_a + \delta_{P^C}(s_p) \sum_{s'_p \in \mathcal{S}_P} \frac{1}{\delta_{P^C}(s'_p)} \frac{\pi_{P^C}(s'_p)}{\pi_{P^C}(s_p)} \lambda_b$$

which is an identity. By uniqueness of the steady-state distribution, we can conclude the proof. □

7.5 Conclusion

This work has addressed the problem of the compositional stationary analysis of PIOAs in a similar fashion to what has been done for the transient in [102]. We have derived a product-form theorem for PIOA. Since we have enlightened the strong relations between the stochastic automata in the style of SANs and the PIOAs, it is interesting to address the problem of relating Theorem 7.4 with the results known for the stochastic counterpart. It is important to notice that the product-form that appears in our theorem is the solution of a DTMC and is *not* equal to that of the corresponding SAN. In fact the discretization procedure described in Section 7.3 does not preserve the steady-state distribution of automata. Moreover, given a PIOA, its steady-state distribution is independent of the δs associated with the states (in the same way the stationary distribution of the embedded chain of a CTMC is independent of its residence times). As a consequence if a PIOA is in product-form for instance according to Theorem 7.4, we can easily compute the stationary distributions of all the corresponding stochastic automata which are defined for arbitrary definitions of the δ functions.

Chapter 8

Conclusions

8.1 Future Work

During the developing of the first part of this thesis, we focused our efforts on the study of the "family" of product-forms. We tried to find methods to transform some systems in product-forms into other systems, altering their behaviours (e.g. rates) but still maintaining satisfied the conditions of RCAT or one of its extensions. We found that the conditions are strictly connected each other, when we tried to change the new system in order to satisfy one condition, another slipped out. Our general impression is that these conditions are strictly bound to the group of product-forms which they can detect. Outside this main "RCAT-group", systems in product-forms seem to be sparse and rarely similar to each other. For now, there was no apparent common scheme in these "outliers" product-forms.
If we analyse discrete time, product-forms are even less grouped. We can consider each time slot constant, in which the system can take more than one action, or in relation to a single change of the system composition (e.g. the number of customers in a queue). Our work is focused on the first type of DT product-forms but we think a similar analysis can be done even in the second case.
We observed that different systems in product-form in continuous time have more similarities if they are transformed in discrete time. Often, if they have proportional rates and similar graphs, they would have the same corresponding discrete time system. We hope that this connection could open the path to a better study of the product-forms in discrete time. This because, even if in continuous time they can't be grouped together under the same conditions, maybe in discrete time a theorem similar to RCAT would be both necessary and sufficient to detect these product-form systems.
For now, we analysed just specific cases in continuous time outside the RCAT theorem and its extension and, in discrete time, we consider just the product-form property for the PIOAs, giving sufficient conditions for a product-form composition. We hope that expanding the research to other kinds of discrete time product-forms, we will be able to track some necessary conditions for product-forms.
Moreover, in discrete time if we consider that more actions can be done in each time slot, we could further exploit the "summary property" of the discrete time. In this case, more continuous actions can be merged in a single discrete state. The main drawback is that we have more state transitions in each state when we model a bunch of moving customers (in a similar way to G-Networks). We would have to analyse this problem, in particular if we consider the time slot representing a large amount of time. However, a discrete system could represent a large amount of continuous one and thus represent a sub-group

of CT product-form.

8.2 Contributions

This thesis is mainly focused on stochastic models in product-form and on limitations of RCAT and quasi-reversible product-forms.
With our work, we propose two novel product-form models that can be applied to the analysis of systems with ageing objects. In the first model, the objects have an unbounded age, while in the other there is a maximum threshold for the age. The approach that we propose can be applied also for the analysis of heterogeneous systems, i.e., systems in which some objects have a maximum threshold for the age and some others have not.
Informally, we can say that the peculiarity of these models is that the transitions are not "local" -as in most of product-form models- i.e., they may change the states of all the objects instantaneously and the effect of a partial rejuvenation event depends on the global state of the model.
We show that the CTMC underlying the model of objects with limited ageing can be seen as an aggregation of the chain of the model with unbounded ageing. This allowed us to prove some equivalence results on the expected performance indices. The consequences of these equivalences are important for practical applications especially if the system must maintain a timer for the objects. Indeed, by introducing a maximum age, we avoid having to maintain the timers of all the objects that reach this age thus reducing the computational effort required for monitoring the object ageing. We used as case-study the analysis of a cache with TTL policy and partial rejuvenation of the objects.
The results can be extended in order to include the partial rejuvenation of clusters of objects (instead of all) and more sophisticated interactions among them in a similar fashion to what is considered in [41]. Indeed, our model may be seen as a queueing network with external independent Poisson arrival streams. The queues can only be partially flushed (i.e., the customers in the network are reduced as computed by Algorithm 1) or totally flushes, i.e., the number of customers in a single queue is set to 0. The exponential distribution of the equilibrium distributions stated by Theorem 5.1 allows the introduction of state-independent probabilistic customer routing.
The proposed model could also be further developed with the possibility of a probabilistic insertion of an object in the cache, i.e. when an object is not in the cache and it is requested, it can enter the cache with a probability q or remain outside with probability $1 - q$. Moreover, we aim at overcoming the limitations of the IRM by allowing state-dependent request rates so that the last requested objects will be associated with a higher rate thus incorporating in the analysis the time-locality property of the network traffic [24].

We have also proposed an abstract modelling framework for the quantitative analysis of reversible computations. The main idea is to exploit the time-reversibility property of Markov processes in order to provide a computationally efficient way of deriving the desired performance indices. We have shown that, under some conditions, the proposed approach is suitable to be applied for a compositional formalism based on labelled stochastic automata. As a consequence, the advantages (reduction of time-complexity and improvement of algorithms' numerical stability) of time-reversibility are applicable also for the analysis of the cooperation of automata that are proved to have product-form

steady-state distributions [5].

Finally, this thesis has addressed the problem of the compositional stationary analysis of PIOAs in a similar fashion of what has been done for the transient in [102]. We have derived a product-form theorem for PIOA. Since we have enlightened the strong relations between the stochastic automata in the style of SANs and the PIOAs, it is interesting to address the problem of relating Theorem 7.4 with the results known for the stochastic counterpart. It is important to notice that the product-form that appears in our theorem is the solution of a DTMC and is *not* equal to that of the corresponding SAN. In fact the discretisation procedure described in Section 7.3 does not preserve the steady-state distribution of automata. Moreover, given a PIOA, its steady-state distribution is independent of the δs associated with the states (in the same way the stationary distribution of the embedded chain of a CTMC is independent of its residence times). As a consequence if a PIOA is in product-form for instance according to Theorem 7.4, we can easily compute the stationary distributions of all the corresponding stochastic automata which are defined for arbitrary definitions of the δ functions [25].

To summarise, the development of this thesis leads us to the publication of the following papers:

1. **A Product-Form Model for the Analysis of Systems with Aging Objects**, (F. Cavallin, A. Marin, S. Rossi; *Proc. of Int. Conf. MASCOTS 2015; pp. 136-145*) [24];
2. **Applying reversibility theory for the performance evaluation of reversible computations**, (M.S. Balsamo, F. Cavallin, A. Marin, S. Rossi; *Proc. of 23rd Int. Conf. ASMTA 2016; pp. 45-59*) [5];
3. **Product-forms for Probabilistic Input/Output Automata**, (F. Cavallin, A. Marin, S. Rossi; *Proc. of Int. Conf. MASCOTS 2016; pp. 361-366*) [25].

Bibliography

[1] P. V. Afshari, S. C. Bruell, and R. Y. Kain. Modeling a new technique for accessing shared buses. *Proc. of the Computer Network Performance Symp.*, page 4–13, 1982.

[2] I. F. Akyildiz. Exact product form solution for queueing networks with blocking. *IEEE Trans. on Computer*, pages C–36–1:122–125, 1987.

[3] I. F. Akyildiz. Exact analysis of queueing networks with rejection blocking. pages 19–29, 1989.

[4] J. R. Artalejo, A. Economou, and M. J. Lopez-Herrero. The maximum number of infected individuals in SIS epidemic models: computational techniques and quasistationary distributions. *J. of Comput. Appl. Math.*, pages 233:2563–2574, 2010.

[5] M.S. Balsamo, F. Cavallin, A. Marin, and S. Rossi. Applying reversibility theory for the performance evaluation of reversible computations. *Proc. of Int. Conf. ASMTA 2016*, pages 45–59, 2016.

[6] S. Balsamo, P. G. Harrison, and A. Marin. A unifying approach to product-forms in networks with finite capacity constraints. pages 25–36, 14-18 June 2010.

[7] S. Balsamo and A. Marin. Performance engineering with product-form models: efficient solutions and applications. *Proc. of ICPE 2011*, page 437–448, 2011.

[8] S. Balsamo and A. Marin. Separable solutions for Markov processes in random environments. *European Journal OF Operational Research*, pages 391–403, 2013.

[9] S. Balsamo, V. De Nitto Persone', , and R. Onvural. Analysis of queueing networks with blocking. *Kluwer Academic Publishers*, 2001.

[10] S. Balsamo, P.G.Harrison, and A.Marin. Methodological construction of product-form stochastic Petri-nets for performance evaluation. *J.Syst.Softw*, page 85(7): 1520–1539, 2012.

[11] E. Barbierato, G.-L. Dei Rossi, M. Gribaudo, M. Iacono, and A. Marin. Exploiting product forms solution techniques in multiformalism modeling. *Electr. Notes Theor. Comput. Sci.*, pages 296:61–77, 2013.

[12] A. D. Barbour. Quasi-stationary distributions in Markov population processes. *Adv. Appl. Prob.*, pages 8:296–314, 1976.

[13] E. Bartocci, F. Corradini, M.R. Di Berardini, E. Entcheva, S.A Smolka, and R. Grosu. Modeling and simulation of cardiac tissue using hybrid I/O automata. *Theoretical Compututer Science*, 410(33-34):3149–3165, 2009.

[14] 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.

[15] D.S. Berger, P. Gland, S. Singla, , and F. Ciucu. Exact analysis of TTL cache networks. *Perf. Eval.*, page 79:2–23, 2014.

[16] M. Bernardo and R. Gorrieri. A tutorial on EMPA: A theory of concurrent processes with nondeterminism, priorities, probabilities and time. *Theoretical Computer Science*, 202:1–54, 1998.

[17] R. Block and B. van Houdt. Spatial fairness in multi-channel CSMA line networks. *Proc. of Valuetools 2014*, page 1–8, 2014.

[18] R. Boucherie. A characterisation of independence for competing Markov chains with applications to stochastic Petri nets. *IEEE Trans.Softw.Eng*, page 20(7):536–544, 1994.

[19] R. Boucherie and N. M. van Dijk. Product-form queueing networks with state dependent multiple job transitions. *Advances in Applied Prob.*, page 23:152–187, 1991.

[20] J. Y. Le Boudec. A BCMP extension to multiserver stations with concurrent classes of customers. *SIGMETRICS '86/PERFORMANCE '86: Proc. of the 1986 ACM SIGMETRICS Int. Conf. on Computer performance modelling, measurement and evaluation*, page 78–91, 1986.

[21] M. Bugliesi, L. Gallina, S. Hamadou, A. Marin, and S. Rossi. Behavioural equivalences and interference metrics for mobile ad-hoc networks. *Performance Evaluation*, page 73:41–72, 2014.

[22] P. J. Burke. The output of a queueing system. *Operations Research*, 4(6):699–704, 1956.

[23] G. Casale and E. Smirni. KPC-toolbox: fitting Markovian arrival processes and phase-type distributions with MATLAB. *SIGMETRICS Performance Evaluation Review*, page 39(4):47, 2012.

[24] F. Cavallin, A. Marin, and S. Rossi. A product-form model for the analysis of systems with aging objects. *Proc. of Int. Conf. MASCOTS 2015*, pages 136–145, 2015.

[25] F. Cavallin, A. Marin, and S. Rossi. Product-forms for probabilistic input/output automata. *Proc. of Int. Conf. MASCOTS 2016*, pages 361–366, 2016.

[26] K. M. Chandy, Jr.J.H. Howard, and D.F. Towsley. Product form and local balance in queueing networks. *J. ACM*, page 24(2):250–263, 1977.

[27] X. Chao, M. Miyazawa, and M. Pinedo. *Queueing networks: customers, signals and product form solutions*. Wiley, 1999.

[28] G. Clark and J. Hillston. Product form solution for an insensitive stochastic process algebra structure. *Perform. Eval.*, pages 50(2): 129–151, 2002.

[29] J. W. Cohen. *The single server queue*. Wiley-Interscience, 1969.

[30] A. Dan and D. Towsley. An approximate analysis of the LRU and FIFO buffer replacement schemes. *Proc. of SIGMETRICS*, page 143–152, 1990.

[31] V. Danos and J. Krivine. Reversible communicating systems. *Proc. of Int. Conf. on Concurrency Theory (CONCUR)*, pages 292–307, 2004.

[32] E. De Souza e Silva and R. R. Muntz. *Stochastic Analysis of Computer and Communication Systems*, chapter Queueing Networks: Solutions and Applications, pages 319–399. H. Takagi Ed., North Holland, 1990.

[33] P. Flajolet, D. Gardy, and L. Thimonier. Birthday paradox, coupon collectors, caching algorithms and self-organizing search. *Discrete Applied Mathematics*, page 39:207–229, 1992.

[34] N.C. Fofack, P. Nain, G. Neglia, and D. Towsley. Analysis of TTL-based cache networks. *Proc. of VALUETOOLS 2012*, pages 1–10, 2012.

[35] N.C. Fofack, P. Nain, G. Neglia, and D. Towsley. Performance evaluation of hierarchical TTL-based cache networks. *Computer Networks*, page 65:212–231, 2014.

[36] J. Fourneau and E. Gelenbe. G-networks with resets. *Perform. Eval.*, pages 49(1–4):179–191, 2002.

[37] J. Fourneau, E. Gelenbe, and R. Suros. G-networks with multiple class negative and positive customers. *Proc. of MASCOTS '94*, page 30–34, 1994.

[38] J. Fourneau and D. Verchere. G-networks with triggered batch state-dependent movement. *Proc. of MASCOTS '95*, page 33–37, 1995.

[39] J.-M. Fourneau. Discrete time Markov chains competing over resources: Product form steady-state distribution. In *Proc of the 5th International Conference on the Quantitative Evaluaiton of Systems (QEST'08)*, pages 147–156, 2008.

[40] J.-M. Fourneau. Collaboration of discrete-time Markov chains: Tensor and product form. *Performance Evaluation*, 67(9):779–796, 2010.

[41] J.-M. Fourneau, L. Kloul, and D. Verchre. Multiple class G-networks with list-oriented deletions. *European J. of Operational Research*, page 126(2):250 – 272, 2000.

[42] J. M. Fourneau and F. Quessette. Computing the steady-state distribution of g-networks with synchronized partial flushing. *ISCIS, 21th International Symposium*, page 887–896, 2006.

[43] J.M. Fourneau and E. Gelenbe. Random neural networks with multiple classes of signals. *Neural Computation*, 11(4):953–963, 1999.

[44] L. Gallina, S. Hamadou, A. Marin, , and S. Rossi. A probabilistic energy-aware model for mobile ad-hoc networks. *Proc. of ASMTA'11*, 6751 of LNCS:316–330, 2011.

[45] E. Gelenbe. A unified approach to the evaluation of a class of replacement algorithms. *IEEE Trans. on Computers*, pages C–22(6):611–618, 1973.

[46] E. Gelenbe. Random neural networks with negative and positive signals and product form solution. *Neural Computation*, 1(4):502–510, 1989.

[47] E. Gelenbe. Non-linear resolvents for very large linear system. *Proc. of the Institute of Math. and its applications Workshop on Numerical Methods in Markov Chains*, 1991.

[48] E. Gelenbe. Product form networks with negative and positive customers. *Journal of Applied Prob.*, page 28(3):656–663, 1991.

[49] E. Gelenbe. G-networks with signals and batch removal. *Probability in the Engineering and Informational Sciences*, pages 7(3), 335–342, 1993.

[50] E. Gelenbe. G-networks with triggered customer movement. *Journal of Applied Probability*, 1993.

[51] E. Gelenbe. G-networks: a unifying model for neural and queueing networks. *Annals of Operations Research*, page 48(5):433–461, 1994.

[52] E. Gelenbe, P. Glynn, and K. Sigman. Queues with negative arrivals. *Journal of Applied Probability*, pages 28: 245–250, 1991.

[53] E. Gelenbe and M. Hernandez. Optimum checkpoints with age dependent failures. *Acta Inf.*, page 27(6):519–531, 1990.

[54] E. Gelenbe and I. Mitrani. Analysis and syntesis of computer systems. *London and New York: Academic Press*, 1980.

[55] E. Gelenbe and R. Schassberger. Stability of g-networks. *Probability in the Engineering and informational Sciences*, pages 6: 271–276, 1992.

[56] E. Gelenbe, S. K. Tripathi, and D. Finkeland. Load sharing in distributed systems with failures. *Acta Inf.*, page 25(6):677–689, 1988.

[57] A. Ghodsi, S. Shenker, T. Koponen, A. Singla, B. Raghavan, and J. Wilcox. Information-centric networking: Seeing the forest for the trees. *Proc. of the 10th ACM Workshop on Hot Topics in Networks, HotNets-X*, page 1:1–1:6, 2011.

[58] W. J. Gordon and G. F. Newell. Cyclic queueing networks with exponential servers. *Operations Research*, page 15(2):254–265, 1967.

[59] W. J. Gordon and G. F. Newell. Cyclic queueing networks with restricted length queues. *Operations Research*, page 15(2):266–277, 1967.

[60] P. G. Harrison. Reversed processes, product forms, non-product forms and a new proof of the BCMP theorem. *Int. Conf. on the Numerical Solution of Markov Chains (NSMC 2003)*, page 289–304, 2003.

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

[62] P. G. Harrison. Compositional reversed Markov processes, with applications to G-networks. *Perf.Eval*, page 57(3):379–408, 2004.

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

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

[65] P. G. Harrison and C. M. Llado'. Hierarchically constructed Petri-nets and product-forms. *5th International ICST Conference on Performance Evaluation Methodologies and Tools Communications, VALUETOOLS '11*, page 101–110, 2011.

[66] P. G. Harrison and A. Marin. Product-forms in multi-way synchronizations. *Comput. J.*, page 57(11):1693–1710, 2014.

[67] W. Henderson and P. Taylor. Product form in networks of queues with batch arrivals and batch services. *Queuing Systems*, page 6:71–88, 1990.

[68] W. Henderson and P. Taylor. Some new results on queueing networks with batch movements. *Journal of Applied Prob.*, page 28:409–421, 1990.

[69] H. Hermanns, U. Herzog, and J. P. Katoen. Process algebra for performance evaluation. *Theoretical Computer Science*, 274(1-2):43–87, 2002.

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

[71] J. Hillston, A. Marin, C. Piazza, and S. Rossi. Contextual lumpability. *Proc. of VALUETOOLS'13*, page 194–203, 2013.

[72] J. Hillston and N. Thomas. A syntactical analysis of reversible PEPA models. *Proc. 6th Process Algebra and Performance Modelling Workshop*, 1998.

[73] J. Hillston and N. Thomas. Product form solution for a class of PEPA models. *Performance Evaluation*, 35:171–192, 1999.

[74] J. R. Jackson. Jobshop-like queueing systems. *Management Science*, 10:131–142, 1963.

[75] K. Kant. Introduction to computer system performance evaluation. *McGrawHill*, 1992.

[76] F. Kelly. Reversibility and stochastic networks. *Wiley*, 1979.

[77] W. F. King. Analysis of paging algorithms. *Proc. of IFIP congress*, page 485–490, 1971.

[78] L. Kleinrock. *Queueing Systems*, volume 1 (Theory). John Wiley and Sons, 1975.

[79] S. S. Lam. Queueing networks with capacity constraints. *IBM Journal of Res. and Dev.*, page 21(4):370–378, 1977.

[80] I. Lanese, C. Antares Mezzina, and F. Tiezzi. Causal-consistent reversibility. *Bulletin of the EATCS*, page 114, 2014.

[81] S. S. Lavenberg. *Computer Performance Modeling Handbook*. Academic Press, New York, 1983.

[82] N. A. Lynch and M. Tuttle. Hierarchical completeness proofs for distributed algorithms. *Proceedings of the 6th Annual ACM Symposium on Principles of Distributed Computing*, 1987.

[83] A. Marin. Product-form in G-networks. *Probability in the Engineering and Informational Sciences*, 2016.

[84] A. Marin, S. Balsamo, and P. G. Harrison. Analysis of stochastic Petri nets with signals. *Perform. Eval.*, 69(11):551–572, 2012.

[85] A. Marin and S. Rossi. Autoreversibility: exploiting symmetries in Markov chains. *Proc. of MASCOTS'13*, page 151–160, 2013.

[86] A. Marin and S. Rossi. On discrete time reversibility modulo state renaming and its applications. *Proc. of Valuetools 2014*, page 151–160, 2014.

[87] A. Marin and S. Rossi. On the relations between lumpability and reversibility. *Proc. of MASCOTS'14*, page 427–432, 2014.

[88] A. Marin and S. Rossi. Lumping-based equivalences in Markovian automata and applications to product-form analyses. *Proc. of the 12th International Conference on Quantitative Evaluation of SysTems (QEST'15)*, 9259 of LNCS:160–175, 2015.

[89] A. Marin and S. Rossi. Quantitative analysis of concurrent reversible computations. *Proc. of FORMATS'15*, 9268 of LNCS:206–221, 2015.

[90] A. Marin and M.G. Vigliotti. A general result for deriving product-form solutions of Markovian models. pages 165–176, 2010.

[91] V. Martina, M. Garetto, and E. Leonardi. A unified approach to the performance analysis of caching systems. *Proc. of INFOCOM 2014*, page 2040–2048, 2014.

[92] B. Melamed and W. Whitt. On arrivals that see time average. *Operations Research*, page 38(1), 1990.

[93] A. Meszaros, J. Papp, and M. Telek. Fitting traffic traces with discrete canonical phase type distributions and Markov arrival processes. *Applied Mathematics and Computer Science*, pages 24(3):453–470, 2014.

[94] R. R. Muntz. Poisson departure processes and queueing networks. *Technical Report IBM Research Report RC4145*, 1972.

[95] A. S. Noetzel. A generalized queueing discipline for product form network solutions. *J. ACM*, page 26(4):779–793, 1979.

[96] K.S. Perumalla. Introduction to reversible computing. *CRC Press*, 2013.

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

[98] S. M. Ross. Stochastic processes. *John Wiley and Sons, 2nd edition*, 1996.

[99] C. H. Sauer and K. M. Chandy. *Computer Systems performance modeling*. Prentice-Hall, Englewood Cliffs, 1981.

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

[101] S.A. Smolka, S.H. Wu, and E. W. Stark. Composition and behaviors of probabilistic I/O automata. *Theoretical Computer Science*, 176:1–38, 1997.

[102] E.W. Stark and S.A. Smolka. Compositional analysis of expected delays in networks of probabilistic I/O automata. In *Proc. of the 13th Annual IEEE Symposium on Logic in Computer Science (LICS'98)*, pages 466–477, 1998.

[103] W. J. Stewart. Introduction to the numerical solution of Markov chains. *Princeton University Press*, 1994.

[104] W. J. Stewart. *Probability, Markov Chains, Queues, and Simulation*. Princeton University Press, UK, 2009.

[105] H. M. Taylor and S. Karlin. *An Introduction To Stochastic Modeling*. Academic Press, 3rd edition, 1998.

[106] D. Towsley. Queuing network models with state-dependent routing. *J. ACM*, page 27(2):323–337, 1980.

[107] S. Trivedi. *Probability and statistics with reliability, queuing and computer science applications*, chapter 8. Wiley-Interscience, second edition, 2002.

[108] N. Tsukada, R. Hirade, , and N. Miyoshi. Fluid limit analysis of FIFO and RR caching for independent reference models. *Perf. Eval.*, page 69:403–412, 2012.

[109] N. van Dijk. Queueing networks and product forms. *John Wiley*, 1993.

[110] P. Whittle. Systems in stochastic equilibrium. *John Wiley & Sons Ltd.*, 1986.

[111] T. Yokoyama and R. Gluck. A reversible programming language and its invertible self-interpreter. *In Proc. of the 2007 ACM SIGPLAN Symposium on Partial Evaluation and Semantics-based Program Manipulation*, pages 144–153, 2007.