Doctoral School MSTII University of Grenoble

PhD Thesis

to obtain the title of
PhD of Science of the University of Grenoble

Specialty : Computer Science

Nicolas Coste

Towards Performance Prediction of Compositional Models in GALS Designs

Thesis supervised by Etienne Lantreibecq and Wendelin Serwe

Defended on June, 24th 2010

JURY

Pr. Brigitte	Plateau	President
Pr. Joost-Pieter	Katoen	Reviewer
Dr. Robert	de Simone	Reviewer
Dr. Hubert	Garavel	Supervisor
Dr. Wendelin	Serwe	Supervisor
Pr. Holger	Hermanns	Examinator
Mr. Etienne	Lantreibecq	Examinator

Acknowledgments

I would like to apologize to people who do not understand French, but I think that I cannot write this acknowledgments section better than in my native language. For people cited here and who do not speak French, I would be glad to thank them in person.

Je tiens à remercier en tout premier lieu les deux personnes ayant encadré cette thèse.

Wendelin Serwe, qui a dirigé cette thèse tout au long de ces trois années, a toujours été très disponible et de bon conseil, en particulier lorsqu'il s'agissait de preuves mathématiques, pour lesquelles j'aurais eu des difficultés sans ses explications. De plus Wendelin a fait preuve d'une (très) grande patience dans la relecture et la correction de ce manuscrit, qui ne serait ce qu'il est sans sa contribution.

Etienne Lantreibecq, mon encadrant industriel chez STMicroelectronics, a lui aussi été d'un grand soutien tout au long de cette thèse. Grâce à lui, j'ai pu garder en tête que mon travail ne devait pas être que théorique, mais aussi exploitable dans le cadre d'études de cas concrètes. Ma proximité avec Etienne, du fait de notre bureau commun, m'a aussi permis de profiter maintes fois de son expérience et je ne saurais citer tout ce que j'ai pu apprendre à ses côtés. Malgré tout, je tiens à m'excuser de lui avoir fait subir un certain nombre de ce qu'il pourrait qualifier de "délires scientifiques", que je gribouillais au tableau blanc du bureau. Cela avait au moins le mérite de servir de décoration et plaisait à tous nos visiteurs !

Je suis donc très reconnaissant envers Wendelin et Etienne pour leur encadrement sans faille et leur grande présence au cours de cette thèse.

En second lieu, je souhaite remercier Holger Hermanns, qui a lui aussi été très présent durant ces trois années. Holger a, sans aucun doute, joué le rôle de catalyseur tout au long de cette thèse, et, travailler avec lui a été une expérience des plus enrichissantes. Holger n'a jamais été à cours d'idées pour m'orienter sur des pistes qui se sont avérées souvent, pour ne pas dire tout le temps, très judicieuses. De plus, sa disponibilité, son soutien, ses qualités d'écoute, et sa sympathie ont rendu nos échanges des plus agréables, ce pour quoi je lui en suis extrêmement reconnaissant.

J'aimerais ensuite remercier les membres de mon jury de thèse qui m'ont fait le plaisir d'évaluer ce travail en qualité de rapporteurs : M. Joost-Pieter Katoen et M. Robert de Simone ; ainsi que Mme Brigitte Plateau qui a accepté de présider mon jury.

Un grand merci à Hubert Garavel qui m'a accueilli au sein de l'équipe VASY et à l'ensemble de ses membres pour leur sympathie. Un remerciement spécial à l'ensemble des ingénieurs experts de VASY, avec lesquels j'ai partagé de très bons moments lors des repas et des pauses café. A leur côté, j'ai pu m'initier à de nombreuses activités, notamment la lecture d'un grand journal (d'information ?) qu'est le Dauphiné Libéré et à l'art du pliage de papier !

Je tiens à remercier l'ensemble des personnes avec lesquelles j'ai pu travailler chez STMi-

croelectronics au sein du laboratoire AST pour leur accueil et leur sympathie. En particulier je souhaite remercier mon chef Richard Hersemeule, pour son soutien et l'intérêt qu'il a porté à ces travaux et l'ambiance qu'il a instauré au sein de l'équipe, ainsi que pour bon nombre d'autres raisons qui sont certainement hors de propos ici. Si je n'ai qu'une chose à dire à Richard, c'est "Merci Chef !".

Merci aussi à toutes les personnes qui ont été impliquées dans le projet MULTIVAL. J'ai toujours trouvé que les échanges lors des réunions trimestrielles étaient intéressants et fructueux. Yvain Thonnard a sans doute sa part de responsabilité quant à l'orientation qu'a pu prendre cette thèse, lorsqu'à une réunion MULTIVAL de décembre 2007, il a émis l'idée : "pourquoi ne pas s'orienter vers une approche à temps discret ?", et je l'en remercie.

Je souhaite remercier Jérôme Ermont, qui, à la fin de mon cursus ingénieur, m'a éclairé quant à la finalité d'une thèse, ce qui a eu une influence certaine sur mon choix. Parmi toutes les raisons que Jérôme a pu me citer, si il en est une que je souhaite souligner, et sur laquelle j'ai maintenant ma propre idée, c'est la satisfaction personnelle que la réalisation d'une thèse peut procurer. En revanche, je ne crois pas me souvenir qu'il ait un jour mentionné ce qu'est la phase de rédaction d'une thèse...

Je tiens à remercier l'ensemble de mes amis pour leur affection. Une mention spéciale à ceux de la PDB dont j'immortalise ici le nom, le faisant ainsi passer à la postérité.

Je tiens à remercier ma famille pour son soutien, sa présence et son affection dans tous les moments de ma vie, et notamment lorsque j'en avais besoin. J'aimerais ici leur témoigner de ma gratitude pour tout ce qu'ils m'ont apporté.

Cette thèse, ainsi que le parcours la précédant, témoignent sans aucun doute de l'amour et de l'enfance heureuse que mes parents m'ont offert. Pour cela, je leur en suis infiniment reconnaissant. Mon engagement dans des travaux de recherche témoigne très certainement de la curiosité des choses que mon père m'a transmise et n'avais de cesse d'entretenir tout au long mon enfance.

Finalement, un grand merci à ma compagne Sandrine, pour sa patience, son soutien et sa compréhension tout au long de cette thèse, et au-delà. Sandrine m'a grandement facilité la vie durant les derniers mois de rédaction, et je tâcherai d'être au moins autant à la hauteur lors de ses prochains examens.

Thesis prepared in the VASY team of INRIA Rhône-Alpes and in the AST Laboratory of STMicroelectronics Grenoble, under the supervision of Hubert Garavel, Etienne Lantreibecq and Wendelin Serwe.

VASY Team, INRIA Rhône-Alpes,
Inovallée, 655, avenue de l'Europe,
Montbonnot, 38334 Saint Ismier Cedex,
France

AST Laboratory, STMicroelectronics,
12, rue Jules Horowitz,
38019 Grenoble Cedex,
France

This document was written with GNU/Emacs *and typeset with* LaTeX *using the additional font package* kpfonts *. Figures have been produced with* xfig, openOffice.org *and* gnuplot *and converted to the .fig format to be inserted in the document.*

Abstract

Validation, comprising functional verification and performance evaluation, is critical for complex hardware designs. Indeed, due to the high level of parallelism in modern designs, a functionally verified design may not meet its performance specifications. In addition, the later a design error is identified, the greater its cost. Thus, validation of designs should start as early as possible.

This thesis proposes a compositional modeling framework, taking into account functional and time aspects of hardware systems, and defines a performance evaluation approach to analyze constructed models.

The modeling framework, called *Interactive Probabilistic Chain* (IPC), is a discrete-time process algebra, representing delays as probabilistic phase type distributions. We defined a branching bisimulation and proved that it is a congruence with respect to parallel composition, a crucial property for compositional modeling. IPCs can be considered as a transposition of Interactive Markov Chains in a discrete-time setting, allowing a precise and compact modeling of fixed hardware delays.

For performance evaluation, a fully specified IPC is transformed, assuming urgency of actions, into a discrete-time Markov chain that can then be analyzed. Additionally, we defined a performance measure, called *latency*, and provided an algorithm to compute its long-run average distribution.

The modeling approach and the computation of latency distributions have been implemented in a toolchain relying on the CADP toolbox. Using this toolchain, we studied communication aspects of an industrial hardware design, the xSTREAM architecture, developed at STMicroelectronics.

Keywords: *performance evaluation, Interactive Probabilistic Chain (IPC), process algebra, Markov chain, latency, hardware architectures.*

Contents

1 Introduction **1**

1.1 Contributions 7

1.2 Outline 8

2 Performance Measures on Markov Chains **11**

2.1 General Definitions of Markov Chains 11

2.1.1 Markov Process 11

2.1.2 Discrete-Time Markov Chain 12

2.1.3 Continuous-Time Markov Chain 18

2.1.4 Steady-State and Transient Probabilities 23

2.2 Annotated Markov Chains 26

2.2.1 Definitions 26

2.2.2 Performance Measures on AMCs 28

2.2.3 Equivalences on AMCs 29

2.3 Semi-Markov Chains 32

2.4 Phase-Type Distributions 33

2.5 Discussion 33

3 Distribution of Latency in a DTMC **35**

3.1 Preliminaries 35

3.2 Latency Definition 36

3.3 Computation of the Latency Distribution 41

3.4 Latency Distribution in Practice 43

3.4.1 Pop Operation Latency 45

3.4.2 Queue End-to-End Latency 46

3.4.3 Discussion 47

4 Interactive Markov Chains **49**

4.1 Interactive Markov Chains 49

4.1.1 Definition 49

4.1.2 Properties of IMCs . . . 52
4.1.3 Markovian Strong and Branching Bisimulations . . . 53
4.1.4 Congruence Property of Markovian Branching Bisimulation . . . 56
4.2 IMCs in Practice . . . 56
4.2.1 Modeling and Analysis of Systems with IMCs . . . 56
4.2.2 Accuracy of CTMC Performance Analysis . . . 57
4.2.3 Bounding the Error on Performance Measures: Hardware Constant Delays Example . . . 58
4.3 Discussion . . . 62

5 Interactive Probabilistic Chains **67**
5.1 Interactive Probabilistic Chains . . . 67
5.1.1 Definition . . . 67
5.1.2 Properties of IPCs . . . 73
5.1.3 Probabilistic Strong and Branching Bisimulations . . . 74
5.1.4 Congruence Property of Probabilistic Branching Bisimulation . . . 77
5.2 From IPCs to DTAMCs . . . 77
5.2.1 Alternating IPC . . . 78
5.2.2 Dealing With Nondeterminism in IPCs . . . 80
5.2.3 Time Deterministic and Deterministic IPC . . . 82
5.2.4 IPC to DTAMC . . . 84
5.2.5 Performance Measures Preservation . . . 86
5.2.6 DTAMC to SMC . . . 86
5.3 IPCs in Practice: Application to a FIFO Queue . . . 87
5.4 Relating IPCs to IMCs . . . 90
5.5 Discussion . . . 95

6 Implementation **97**
6.1 IPC Modeling . . . 97
6.1.1 Storing an IPC Using the CADP Toolbox . . . 97
6.1.2 Using LOTOS to Express IPCs . . . 99
6.1.3 Compositional Approach to Generate IPCs . . . 100
6.1.4 Constraint-Oriented Specification of IPCs . . . 100
6.1.5 Parallel Composition of IPCs . . . 107
6.1.6 Compositional IPC Modeling . . . 110
6.2 Computation of the Latency Distribution . . . 110
6.3 Discussion . . . 115

7 Industrial Case-Study: The xSTREAM Architecture **117**
7.1 Presentation of the xSTREAM Architecture . . . 117

7.1.1 Stream-Oriented Programming Model . . . 117
7.1.2 Design of the xSTream architecture . . . 118
7.1.3 Pop queues . . . 120
7.1.4 Network-on-Chip . . . 121
7.1.5 Credit Protocol . . . 122
7.2 Functional Models of the xSTream Architecture . . . 123
7.2.1 Modeling choices . . . 123
7.2.2 Push Queue Model . . . 124
7.2.3 Pop Queue Model . . . 125
7.2.4 Simple Virtual Queue . . . 126
7.2.5 Complex Virtual Queue . . . 128
7.2.6 Two Parallel Virtual Queues Sharing Resources on the NoC . . . 132
7.3 Performance Measures for the xSTream architecture . . . 134
7.3.1 Study of a Simple Virtual Queue . . . 134
7.3.2 Study of Two Parallel Virtual Queues Sharing Resources on the NoC . . . 139

8 Related Work 145
8.1 Process Algebras Dealing With Time and/or Probabilities . . . 145
8.1.1 Time Models . . . 146
8.1.2 Probabilistic Models . . . 147
8.1.3 Alternating Discrete-Time and Probabilistic Models . . . 148
8.2 Bisimulations . . . 150
8.3 Analysis of Interactive Probabilistic Chains . . . 152

9 Conclusion 155

A Proof of Lemma 5.5 159

B Proof of Theorem 5.1 161

C Proof of Lemma 5.8 165
C.1 Scheduling of Bisimilar Urgency-Cut tdIPCs . . . 165
C.2 DTAMCs Associated to Branching Bisimilar dIPCs . . . 169
C.3 Associated DTAMCs of Branching Bisimilar tdIPCs . . . 171

D Proof of lemma 5.9 173

E LOTOS models 181
E.1 Data Type Libraries . . . 181
E.1.1 Queue Size . . . 181
E.1.2 Queue Elements . . . 182

E.1.3 Queue 182
E.1.4 Push Queue Identifiers 183
E.1.5 Pop Queue Identifiers 183
E.2 Push Queue 183
E.3 Pop Queue 190
E.4 Network-on-Chip abstraction 196

Chapter 1

Introduction

Electronic embedded multimedia devices, such as mobile phones, GPS, video or music players, electronic diaries, etc., are becoming more and more present in our daily lives. While these devices were rather simple and dedicated to a single function few years ago, nowadays they tend to be increasingly complex, merging several different functions. Mobile phones are the most obvious example of this evolution. It is natural at the present time to have many rather different functions enabled on the same phone device: camera, video recorder, music player, HD-video player, web browser, GPS, diary, games, etc. While a dedicated hardware design was able to ensure the required functions few years ago, at the present time, the combination of several different functions in the same device requires a complex multipurpose hardware design. Unfortunately, the technical requirements for such an embedded design are often conflicting. Computing capabilities have to be combined with power saving, complex functions should be upgradable, etc. In addition, technical requirements may evolve during the lifetime of a device, which is difficult to take into account in the design phase.

The ability to ensure complex tasks in an electronic device is directly linked to its processor performance. The evolution of single processor performance followed Moore's law for the last thirty years: processor performance doubling every two years. Recently, however, certain physical limits (such as frequency and heat dissipation) of semiconductor-based chips have been reached, which has caused a technological breakthrough in designs: Single-processor designs have been replaced by multiprocessor designs. This corresponds to a switch from a mostly synchronous world with little parallelism, to an asynchronous world with a high degree of parallelism. This transition started with designs of processors for personal computers and now happens for embedded device processors. In addition, we are progressively switching to systems where software takes a larger and larger part. This kind of design may benefit of several improvements: increased computation capabilities, fine-grain power management, homogeneity in the fabrication process, etc. Unfortunately, those improvements induce an increased complexity. Indeed, different execution threads may interact or share resources. Maintaining data coherency and preventing a shared resource from being improperly preempted is a key problem. In other words, it makes sense to ensure that the design under development has the ability to perform the targeted tasks. This process, well known by designers for a long time, is called design verification. It occurs during the conception phase of the system, and consists of establishing the correctness of the design, i.e., verifying that the system matches its specification before implementation.

The verification process starts by providing a model of the system, i.e., finding a suitable

representation of its behavior in such a way that it can be analyzed. A classical approach for modeling a system is to represent its behavior by defining a discrete set of states the system may occupy (also called its state space), and by defining the evolution (or transition) function from one state to another. Then, properties extracted from the system specification are verified using the model. The verification process covers two different activities, according to the nature of the properties to be verified: properties linked to the functional behavior of the system or properties linked to the timed behavior of the system.

The first aspect of correctness concerns the functional behavior of the system. It deals with the way the system ensures its task and states, for instance, with properties such as: "is it possible for the system to be definitively blocked" (e.g., two processes needing respectively a resource preempted by the other), "will the system eventually fulfill its task ?", "is the behavior the one we were looking for ?". The study of those functional properties is called functional verification. To perform functional verification, a model depicting the functional behavior of the system is needed. In this kind of model, the transition from one state to another is consequently based on functional information (e.g., an action processed). For the case of an asynchronous system, one may cite Labeled Transition Systems (LTS) as a typical formalism used in the field of functional verification. LTS are directed graphs where vertices represent the states of the modeled system and directed edges, labeled by an action, represent transitions between states. Actions of LTS are considered atomic, i.e., they are taken instantaneously (there is no time representation). An LTS allows to depict the functional behavior of a system: it only depicts the scheduling of its atomic actions, without providing information on the time elapsed between actions. For asynchronous systems, interleaving semantics is used to depict parallelism. For instance, if the LTS depicts two behaviors in parallel, all the possible interleaving of actions of the two behaviors will be represented. LTS is a simple formalism that can be easily manipulated, and is at the base of most of the functional verification techniques.

The second aspect of correctness concerns the performance of the system. It deals with the time spent by the system to ensure its tasks and states, for instance, with properties such as: "does the video-decoder ensure a video rate of 25 frame per second ?", "what is the video buffer size needed for streaming ?". The study of those quantitative properties is called performance evaluation. To deal with performance evaluation, a model depicting the evolution of the system in time is needed. In this kind of model, the transition from one state to another represents the progress of time. One may cite Markov chains as a typical formalism in the field of performance evaluation. Like an LTS, a Markov chain is also a directed graph where vertices represent the states of the modeled system, but directed edges represent a probabilistic evolution, inducing that time progresses, from one state to another. A Markov chain is a simple formalism that can be easily manipulated, and is widely used to estimate performance measures of systems.

Up until few years ago, for processors with a low level of parallelism, correctness was mainly ensured by answering functional verification questions. Indeed, problems concerning performance targets could be generally fixed, under some conditions, by frequency adjustments, or by improving critical paths in the circuit. Performance requirements could be reasonably assumed to be ensured if the system was functionally verified. For this reason, it was common to consider performance only after the hardware system was fully designed and functional correctness ensured.

At the present time, a functionally verified multiprocessor architecture may not reach its required performance specifications. Indeed, due to concurrency, communications may be delayed and latencies appear, directly worsening the system performance. A system non-

compliant with timed specifications may require radical modifications of the architecture to be rendered compliant. Such drastic modifications on fully developed systems are nowadays unthinkable due to prohibitive hardware development costs. Performance evaluation is consequently mandatory in addition to functional verification in the verification process of a system. This implies that performance evaluation needs to be processed sooner, before the first prototypes and even before having precise description of the architecture. Early evaluation will prevent the designers from making choices that would lead to designs that are unable to meet the requirements.

In the field of hardware systems presenting a high level of parallelism, the performance concerning the achievement of a given task no longer relies on the single hardware design, but also depends on all the tasks running concurrently. Unfortunately, depending on the targeted task, performance is of critical importance (e.g., real-time or streaming applications). Performance measures on isolated hardware elements (like throughput of an element) are consequently no longer sufficient to ensure the ability of the system to complete the targeted task. Performance evaluation has to take into account both the hardware design, and the software running on it. We must therefore study the performance of a complex hardware system within a defined application context.

The performance measures of interest in hardware systems may be of very different nature, covering questions concerning applications (e.g., given an application context, a video decoder has to run properly) or hardware elements (e.g., the communication bus has to ensure a precise throughput). We identified three major generic kinds of measures to be considered when studying a complex hardware system:

- **Resource utilization**. These measures provide information concerning the utilization of some elements, and mainly concern hardware elements (e.g., an hardware queue mean occupancy). They can be used to identify under-utilization or over-utilization of some resources and to size them properly.
- **Throughput**. These measures may provide information on hardware elements or on applications. They are of main interest when studying the ability of the system to ensure tasks highly dependent on time.
- **Latency**. These measures may provide information on hardware elements or on applications. We can identify local latencies (e.g., time to process an operation) or end-to-end latencies (e.g., time to pass through a communication element). Due to high-level of parallelism in the studied systems, contentions may appear, worsening latencies. For some applications, this may be annoying and has to be reduced as much as possible.

All these measures of interest concern behaviors on average or in the long run. Indeed, our targeted systems (dedicated to the processing of not critical applications like streaming applications) are not dealing with hard real-time applications, but they have to be able to fulfill well their task on average.

The targeted performance measures depend on delays inherent in the modeled systems. The classical approach taking time into account in models is to consider timers representing those delays. The chosen model of time has to include a method of depicting all the delays present in the modeled system. In the case of a hardware system, one could consider that delays are constant and precisely known. Indeed, simple hardware elements composing the system have generally a deterministic behavior, which presents few variability in time. Nevertheless, modeling an application using precise delays would imply a great accuracy of the model that we would prefer to avoid by considering application patterns instead. Indeed, because targeted

results are long-run or average measures, we would prefer to abstract from a precise behavior in time of an application, and consider a shape of behavior in time. In this case, a probabilistic view of delays is preferred. We will consequently have to consider delays as random variables characterized by their probabilistic distribution.

Example 1.1 illustrates the notion of delays and the targeted performance measure for a very simple hardware element with applications, presenting little parallelism. This example will be used throughout this thesis to illustrate important concepts simply.

Example 1.1. *Consider a small hardware component: a First-In First-Out (FIFO) hardware queue. The operation of inserting (resp. withdrawing) an element into the queue will be called push operation (resp. pop operation). This terminology (push and pop) is classically used for stacks but is inspired here by the terminology of* xSTream *queues (*xSTream *architecture, designed at STMicroelectronics, is presented in chapter 7). Push and pop operations are processed using a hand-shake scheme: the operations are initiated by a request and when they have been processed, a response is sent back. For the push operation the response is just an acknowledgment, while the element at the queue's head is returned for the pop operation. Push and pop operations can be processed either in parallel or sequentially according to the implementation.*

Because we consider a hardware queue, its size is limited and operations are blocking: when a push (resp. pop) operation is initiated by a request and the queue is full (resp. empty), the operation is blocked until there is a free place (resp. an available element). For processes inserting and withdrawing elements in the queue, this behavior corresponds to a busy wait.

For performance evaluation purposes, the FIFO queue is studied in an application context: a producer inserts (push operation) elements and a consumer extracts them (pop operation). This example is depicted in figure 1.1. Considering this FIFO queue and its environment, we can distinguish two hardware-related delays and two software-related delays. The hardware delays correspond to the time physically needed to process operations in the queue. They are called the push operation delay and the pop operation delay. The software delays are linked to the timed behavior of the producer and consumer: how much time is spent between the end of an operation (push or pop response) and the beginning of the next one (push or pop request)? Those delays are called production delay and consumption delay. One can imagine that hardware delays are constant, while software delays are considered as random variables.

Consequently, the system (the queue connected to a producer and a consumer) has five parameters:
- *S_Q: the size of the queue*
- *D_{PUSH}: the push operation delay*
- *D_{POP}: the pop operation delay*
- *D_{PROD}: the production delay*
- *D_{CONS}: the consumption delay*

We define the mean production rate (resp. consumption rate) as the number of elements produced (resp. consumed) on average by time unit. We will say that the queue is well used if the mean consumption rate in the queue is greater or equal to the mean production rate. In practice we generally consider a well-used system, because a system that is not well-used is certainly not desirable: after a transient phase, the queue will always overflow. Studying this hardware queue allows us to investigate several performance results among which:
- *Occupancy of the FIFO queue. This measure provides information to correctly size the queue. Indeed, because it is an hardware queue, a trade-off between the performance (an infinite queue ensures there is never blocked insertions) and the cost due to die area occupancy has to be found. A queue that is often full is probably not large enough, while a queue that is often empty could*

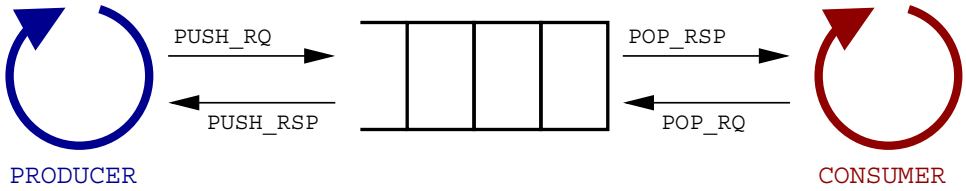


Figure 1.1: A FIFO queue connected to a producer and a consumer

be smaller.

- *Time between an operation request and the next one. This time is interesting for the study of throughput measures concerning push and pop operations.*
- *Time between an operation request and the corresponding response. If this time is much greater than the operation hardware delay, the operation is often blocked. For a push operation, it means that the queue is not able to correctly absorb the traffic. For a pop operation, it means that the consumer is often waiting.*
- *Time for an element to pass through the queue. This is the time between the push request to insert the element in the queue and the pop response withdrawing it from the queue. This time corresponds to the latency induced by the queue in the communication between the producer and the consumer.*

In industry, functional verification and performance evaluation of complex hardware systems are typically currently tackled as two independent tasks, relying on two different models. This approach is the source of several problems: it causes effort and cost redundancy, incoherence between the two models may appear, diverging evolutions in the two models may induce difficulties in their maintenance, etc. At the moment, there is no available industrial methodology that covers both functional verification and performance evaluation of the same model. To remedy this problem, a common model needs to represent both functional evolutions of the system, and time evolutions. Typically, modeling frameworks for functional verification and performance evaluation consider atomic actions depicting the functional behavior, and define timers representing time periods that have to elapse between actions. One may imagine that the transition from one state to another implies at the same time the occurrence of an action and the elapsing of a period of time. In this case, an action is no longer atomic, but takes a certain amount of time. A clear separation between time evolution and functional evolution is preferable, because it is theoretically simpler without limiting the applicability of the approach (a non atomic action can be modeled using two atomic actions). This observation has been illustrated in [NS91] for timed systems.

Given the need to validate complex hardware systems, one can thus propose some guidelines for defining an adequate framework dedicated to functional verification and performance evaluation. In this aim, several questions have to be answered.

The first question arises in the modeling phase of the targeted systems, and concerns the management of the complexity of models, to scale to models with a huge state space. To deal with complexity, engineers are used to develop systems in a compositional and hierarchical approach: complex systems are developed by assembly of several simpler subcomponents. In the field of system modeling, this approach is mandatory for scaling to large systems with a high level of parallelism: the whole system is divided into subcomponents that can be modeled independently, and are then composed to get the final model. This kind of approach implies

that one is able to explicitly express parallelism and interactions between subcomponents of the system to model. In other words, subcomponents are modeled as independent entities evolving in parallel, but interacting through means of synchronization. Such an approach implies that the composition of two models has to remain representable in the chosen modeling framework.

When composing models of subcomponents in parallel using interleaving semantics, the state space of the resulting model might, in the worst case, be the Cartesian product of the subcomponents' state spaces. A high level of parallelism between composed models generally results in an exponential increase of the state space of the resulting model. This leads to the so-called state space explosion problem: the model resulting from a composition is too large to be analyzed (in the sense that it is too large to be numerically manipulated by modern computers). Because complexity of studied systems (and thus size of the models) increases faster than computing capabilities of computers (even considering clusters of computers), one cannot count upon the next more powerful generation of computers to resolve future state-space explosion problems.

An approach to circumvent the state space explosion problem is the use of compositional minimizations using equivalences: some components of the system are replaced by equivalent but smaller ones, i.e., minimized subcomponents (with respect to those equivalences). Those minimizations are defined so as to preserve properties in the model. They rely on the definition of equivalence relations (also called bisimulations) between states with respect to the preserved properties. Given an equivalence relation, a model is minimized finding the smallest one equivalent to it. By iteratively composing subcomponent models [GL01], and minimizing the results, we construct a minimized model of the whole system that should remain equivalent to the one we would have obtained without minimization. In some cases, the approach based on compositions and minimizations may avoid the state space explosion problem, because the intermediate state spaces of compositions are reduced. In this thesis, the term "compositional modeling approach" refers to the construction of a model using iterative composition and minimization.

When a model of the system is available, a second question arises. It concerns the way the model is analyzed to get functional verification and performance evaluation results. One can distinguish two classes of methods for evaluating the correctness of a system: simulation-based methods and exact methods. Simulation-based methods consist in verifying properties or computing measures by considering executions of the model. In contrast, exact methods consist in computing results using a clearly defined mathematical framework. One can mention advantages and drawbacks for the two approaches:

- For simulation-based methods, it is difficult to ensure exact results. Indeed, it would imply that results are computed on an infinite execution in which all the behaviors of the model have been tested. In most cases, simulation-based results are therefore not given as ensured results, but with a confidence index with respect to the time of simulation (i.e., until a criterion of convergence is reached) or the coverage of the set of possible behaviors. Excepted for the evaluation of the confidence index of the results, simulation-based methods remain rather simple techniques.
- Computed results are ensured using exact methods, i.e., if a result is found with formal methods, it is correct and does not need a confidence index. But results using formal methods are obtained at the cost of more complex algorithms.

Actually, the exponentially increasing complexity in hardware systems implies that simula-

tions are taking more and more time to ensure a reasonable confidence index. Formals methods are consequently becoming interesting as simulation-based methods reach their limits [Kur97].

In industry, formal methods are beginning to be used for the functional verification of complex hardware systems [KGN+09, Ber05, Kur97], as a complement of simulation-based methods. Conversely, simulation-based methods are the standard for performance evaluation. However, formal methods are seen as a great aid to providing both functional and performance results for such complex systems.

To summarize: the validation flow of a complex hardware system should ideally first rely on a model covering both functional verification and performance evaluation. This model has to depict both software and hardware parts of the system. Then, the model should be constructed following a compositional approach. Finally, it should be analyzed using formal methods. The expected results are the verification of functional properties, and performance measures in terms of latencies, throughput and resource utilization.

1.1 Contributions

To tackle problems described in the previous section, we mainly focus on the modeling of hardware systems and how to formally proceed with performance analysis. Consequently, the contribution of this thesis is twofold:

- Our first contribution is the definition of the Interactive Probabilistic Chains (IPC), which allow to model systems with both functional information and time information. In IPCs, time is considered to be discrete and probabilistically distributed, i.e., a probabilistic decision is authorized when time progresses. By means of discrete phase-type distributions, any kind of discrete-time distributed delay can be modeled with IPCs.
  The operational semantics of IPCs strictly separates functional aspects and time aspects. This simplifies the definition of parallel composition. In particular, it is not necessary to deal with synchronization of probabilistic distributions. Consequently, for IPCs, the parallelism of functional behavior is defined asynchronously (according to the usual interleaving semantics), while a synchronous semantics is used when dealing with parallelism of timed behaviors. In addition, to authorize a compositional modeling approach with intermediate minimizations, we define a probabilistic branching bisimulation equivalence of IPCs that we prove to be a congruence with respect to the parallel composition operator.
  By their nature, IPCs can be used to tackle both functional verification aspects and performance evaluation aspects. Indeed, an IPC can be minimized to obtain an LTS or transformed into a discrete-time Markov chain. The transformation of an IPC into a Markov chain relies on an urgency assumption, i.e., actions occur as soon as they are allowed to be taken.
  We implemented the IPC modeling approach by reusing existing tools and APIs of the CADP toolbox [GLMS07], and by developing additional tools to deal with the synchronous semantics of time in IPCs. We applied this implementation on to model the communication aspects of an industrial hardware design, the xSTREAM architecture, an adequate case-study to evaluate the scalability of the IPC modeling approach.
- Our second contribution is the formal definition of a performance measure, the latency, which characterizes the time elapsed between two particular set of states in a discrete-time Markov chain. We show that the long-run average distribution of a latency can

be expressed using classical steady-state and transient analysis techniques on Markov chains. We show how this latency can be used for performance study of an IPC (namely, by considering its transformation into in a Markov chain).
We provide an implementation of the computation of the long-run average distribution of a latency on an IPC, and we compute some performance figures on the IPC models of the xSTream architecture. On these models, we illustrated that we are able to cover two out of the three measures of interest we identified for hardware systems, namely the throughput and latencies.

1.2 Outline

In chapter 2, we present, in discrete-time and continuous-time settings, those aspects of the Markov chain formalism that are necessary in the rest of the thesis. In particular, we recall well-established techniques to obtain performance results from continuous-time and discrete-time Markov chains. Then, we introduce an extension of Markov chains, called annotated Markov chains that authorize to associate functional information to the states of a Markov chain. Finally, we present the related formalisms of phase-type distributions and semi-Markov chains.

In chapter 3, we introduce a performance measure on discrete-time annotated Markov chains: the latency, as a random variable relying on the construction of particular set of states that are identified by the actions they can perform. We focus on the computation of the average distribution of a latency on the long run. Although the definition of the long-run average distribution of a latency relies on notions belonging to measure theory, we show that its computation is possible based on steady-state and transient-state analysis of Markov chains.

Chapter 4 is dedicated to the presentation of the Interactive Markov Chains (IMC), an adequate formalism to tackle both functional verification and performance evaluation. Indeed, we intially planned to use IMCs to study hardware systems. Firstly, we recall the formal definition of the IMC formalism. Then, we shortly overview a practical methodology to apply this formalism to the modeling of hardware systems. From IMC models, we show how to get performance results by considering the performance model underlying IMCs, i.e., continuous-time Markov chains. Finally, we illustrate the problems encountered with the IMC formalism: the approximations of constant delays (and more generally of discrete-time delays) by exponential distributions induces an error on computed performance results, that we are not able to evaluate.

In chapter 5, we introduce the Interactive Probabilistic Chain (IPC) formalism, which is the main contribution of this thesis. IPCs have the same modeling and performance analysis capabilities as IMCs, but they model discrete-time delays exactly, i.e., without inducing approximation errors. Firstly, we define IPCs and provide their operational semantics. Secondly, we focus on their performance analysis. Similarly to IMCs, the underlying performance model of IPCs is a discrete-time Markov chain. We define a transformation of IPCs into annotated Markov chains and show how a latency can be studied through the latency in the annotated Markov chain associated to an IPC. Finally, we compare IMC and IPC semantics and highlight a strong relation between those two formalisms.

In chapter 6, we present a prototype toolchain for studying systems modeled using the IPC formalism. This tool chain is divided in two parts: the first one concerning the modeling of a

system as an IPC, and the second one tackling the performance analysis of this IPC. Currently, our implementation enables the study of long-run average distributions of latencies.

In Chapter 7, we present the application of our toolchain to an industrial case-study, the xSTream architecture. We present some models we studied, together with the obtained results, highlighting the advantages of the IPC methodology for modeling hardware systems. We also show that the latency measure allows to get useful insight concerning the performance of the studied systems.

In chapter 8, we overview existing work related to the main contributions of this thesis, namely IPCs and the latency performance measure on discrete-time Markov chains.

Finally, in chapter 9, we give some concluding remarks concerning the modeling and performance evaluation of hardware systems. We also present some directions concerning further research.

Chapter 2

Performance Measures on Markov Chains

In this chapter we present Markov chains, a formalism commonly used in computer-aided performance evaluation methodologies [GH02, HKNP06, PA91]. Intuitively, a Markov chain is a stochastic process used to depict systems evolving randomly in time. Markov chains have been widely studied in the literature and efficient performance analysis methods exist [Ste94, Hav00]. Indeed, due to their discrete state space, they are well adapted for numerical representations and manipulations. In a Markov chain, time can be considered either discretely or continuously dividing Markov chains into two classes: discrete-time Markov chains (DTMCs) and continuous-time Markov chains (CTMCs). The purpose of this chapter is to present general definitions concerning DTMCs and CTMCs, together with established performance analysis methods. We also introduce a formalism based on Markov chains, the annotated Markov chains (AMCs). Finally, we present phase-type distributions constructed over Markov chains.

2.1 General Definitions of Markov Chains

2.1.1 Markov Process

Stochastic processes are well-studied formalisms to depict systems evolving in time. In a stochastic process, the evolution in time is not deterministic but is described by probability distributions.

Definition 2.1 (Stochastic process)**.** *A* stochastic process X *is a collection* $\{X_t \mid t \in T\}$ *of random variables* X_t *taking values in a set* S*, defined on a given probability space and indexed by the parameter* t*, where* t *varies over some index set* $T \subseteq]-\infty, +\infty[$.

The set T is usually seen as the time parameter called the *time range*. If T is a discrete countable set (usually, $T = \{0, 1, \ldots\}$), the process is called a *discrete-time process*. Otherwise, if T is not countable (usually, $T = \mathbb{R}_+$), the process is called a *continuous-time process*.

$X_t \in S$ denotes the value assumed by the process X at time t. The elements of set S are called *states*, and the set S of all states forms the *state space* of the process. The state space is either discrete or continuous.

A *Markov process* is a stochastic process satisfying the so-called *Markov property*. Informally, this property expresses that the future of a Markov process only depends on its present (the current occupied state) and not on its past (the previously occupied states).

Definition 2.2 (Markov process)**.** *A Markov process is a stochastic process satisfying the Markov property: for any sequence of time points t_0, t_1, …, t_n, t_{n+1} satisfying $t_0 < t_1 < \cdots < t_n < t_{n+1}$ (time points are ordered in time), the Markov property ensures that*

$$Prob\left\{X_{t_{n+1}} = x_{n+1} \mid X_{t_n} = x_n, X_{t_{n-1}} = x_{n-1}, \ldots, X_{t_0} = x_0\right\} = Prob\left\{X_{t_{n+1}} = x_{n+1} \mid X_{t_n} = x_n\right\}$$

For a Markov process, the distribution of X_{t_n} on the state space at time t_n can be determined by the distribution of $X_{t_{n-1}}$ only. Nevertheless, the distribution of X_{t_n} may depend on the time point t_n considered. In the case that the distribution of X_{t_n} is independent from t_n, each time point is a *renewal point*, and the Markov process is said to be *homogeneous*.

Definition 2.3 (Homogeneous Markov process)**.** *A homogeneous Markov process is a stochastic process satisfying the Markov property and*

$$\left(\forall t' > t\right), \quad Prob\left\{X_{t'} = x' \mid X_t = x\right\} = Prob\left\{X_{t'-t} = x' \mid X_0 = x\right\}$$

In the current work, we focus on homogeneous Markov processes defined on a discrete state space. They are called *Markov Chains*.

Definition 2.4. *A Markov chain is a Markov process defined on a discrete state space.*

2.1.2 Discrete-Time Markov Chain

Definition

In this section, we introduce notions concerning DTMCs. A full presentation of DTMCs can be found for instance in [Nor98].

Definition 2.5 (DTMC)**.** *A stochastic process $X = \{X_t \mid t \in T\}$ is a discrete-time Markov chain if X is a discrete-time process and X is a Markov chain.*

In a DTMC, the time increases discretely by *time steps*. Each move from one state to another is processed in one time step and is called a *transition*. We consider the usual time range used for discrete-time stochastic processes, i.e., $T = \mathbb{N}$, which allows to denote a DTMC $X = \{X_n \mid n \in \mathbb{N}\}$.

Let $\{X_n \mid n \in \mathbb{N}\}$ be a homogeneous DTMC defined over the discrete and finite state space $S \subset \mathbb{N}$. For every pair of states $(i, j) \in S^2$, the probability of moving in one time step from state i to state j at time n,

$$p_{ij}(n) = \mathsf{Pr}\{X_{n+1} = j \mid X_n = i\},$$

is called *transition probability at time n* from state i to state j. In a homogeneous DTMC, each time point is a renewal point, i.e.,

$$\left(\forall n \in \mathbb{N}\right) \quad p_{ij}(n) = p_{ij}(0) = \mathsf{Pr}\{X_1 = j \mid X_0 = i\}.$$

Thus, we define $p_{ij} = p_{ij}(0)$ and p_{ij} is simply called the *transition probability* from state i to state j. All probabilities out of a state are assumed to sum up to one, i.e., $\left(\forall i \in S\right), \sum_{j \in S} p_{ij} = 1$. The square matrix $P = [p_{ij}]_{(i,j)\in S^2}$ is called the *transition matrix*.

In a homogeneous DTMC, the sojourn time SJ_i in a state $i \in S$ is the time spent in i before moving to another state. For every state $i \in S$, SJ_i follows a geometric distribution parameterized by the transition probability p_{ii}. Its distribution is defined by:

$$\mathsf{Pr}\{\mathsf{SJ}_i = n\} = p_{ii}^{n-1} \times (1 - p_{ii})$$

The geometric distribution is the only *memoryless* discrete probability distribution, i.e., the remaining time before leaving the state is independent from the time already spent in the state. The memoryless property of the distribution of the sojourn time in a DTMC is a direct consequence of the Markov property.

Consider the vector $\pi(n) = [\pi_i(n)]_{i\in S}$, where for every state $i \in S$, $\pi_i(n) = \mathsf{Pr}\{X_n = i\}$. $\pi(n)$ is called *probability vector* of the DTMC at time n and corresponds to the probability distribution of the DTMC over the state space at time n. This probability vector satisfies $\left(\forall n \in \mathbb{N}\right), \sum_{i \in S} \pi_i(n) = 1$, which simply ensures that, at any time, the probability to be in the state space is 1.

A homogeneous DTMC is fully characterized by its transition matrix and its initial probability vector $\pi(0)$. Indeed, all possible evolutions can be computed from the initial probability vector and the transition matrix of a DTMC. In the rest of the thesis, we limit our study to DTMCs for which a single state is initially occupied. In practice, this assumption is in most cases verified.

Example 2.1. *Consider the hardware FIFO queue presented in example 1.1 with the following parameters in a discrete-time context:*

- *The queue size Q_S is set to one place, $Q_S = 1$*
- *The hardware related delays D_{PUSH} (push operation delay) and D_{POP} (pop operation delay) are respectively set to $D_{PUSH} = 2$ and $D_{POP} = 1$ time steps*
- *The software-related delays D_{PROD} (production delay) and D_{CONS} (consumption delay) are considered probabilistically. D_{PROD} can take either 2 time steps with a probability 0.5 or 3 time steps with a probability 0.5. D_{CONS} can take either 1 time step with a probability 0.5 or 4 time steps with a probability 0.5. We can notice that the FIFO queue is well used: there is, on average, one element produced or consumed every 2.5 time steps.*

Initially, we assume that the queue is empty, and that production and consumption delays have to elapse before first insertions or extractions in the FIFO queue. This system can be modeled using a DTMC X defined over a state space $S = \{s_0, \ldots, s_{29}\}$. The transition matrix $P = \left[p_{s_i s_j}\right]_{(s_i,s_j)\in S^2}$ and the initial probability vector $\pi(0) = \left[\pi_{s_i}(0)\right]_{s_i \in S}$ of X are depicted in figure 2.1.

State properties

We introduce several classifications of states that will be used afterwards. Consider the homogeneous DTMC $\{X_n \mid n \in \mathbb{N}\}$ defined over the discrete and finite state space $S \subset \mathbb{N}$, and characterized by its transition matrix $P = [p_{ij}]_{(i,j)\in S^2}$ and its initial probability vector $\pi(0)$.

$$P = \begin{bmatrix}
0 & \frac{1}{4} & 0 & 0 & 0 & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}$$

$$\pi(0) = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Figure 2.1: Transition matrix and initial probability vector of a 1-place FIFO queue DTMC model

Given a state $i \in S$, the probability ret_i^k of returning to i in k time steps, when starting in i, is defined by:

$$\text{ret}_i^k = \Pr\{X_k = i \mid X_0 = i\}.$$

If any return to i is a multiple of a value $k > 1$, the state i is said to be *k-periodic*. Otherwise, i is said to be *aperiodic*.

Definition 2.6 (periodic state)**.** *a state i is said to be k-periodic ($k > 1$) if:*

$$\left(\forall n \in \mathbb{N}\right), \quad gcd\left\{n \mid ret_i^n > 0\right\} = k$$

A DTMC is said to be aperiodic if all its states are aperiodic.

We also define the probability ret_i of ever returning to state i when starting in i by

$$\text{ret}_i = \sum_{n=1}^{\infty} \Pr\left\{X_n = i \mid X_0 = i \wedge \left(\forall n' < n\right) X_{n'} \neq i\right\}.$$

Definition 2.7 (Transient and recurrent states)**.** *A state $i \in S$ is said to be* recurrent *if, starting in i, the probability ret_i to ever return to i is* 1*. If, starting in i, there is a non-zero probability to never return in i, the state is said to be* transient*.*

Then, we define the mean recurrence time mean_i of state i:

$$\text{mean}_i = \sum_{n=1}^{\infty} n \times \Pr\left\{X_n = i \mid X_0 = i \wedge \left(\forall n' < n\right) X_{n'} \neq i\right\},$$

i.e., the average time needed, starting in i, to return to i.

Definition 2.8 (positive recurrent state)**.** *A state $i \in S$ is said to be* positive recurrent *if, starting in i, the mean recurrence time mean_i is finite, i.e., $\text{mean}_i < +\infty$.*

Definition 2.9 (ergodic state)**.** *a state $i \in S$ is said to be* ergodic*, if it is aperiodic and positive recurrent.*

A DTMC is also said to be ergodic if all its states are ergodic.

Definition 2.10 (absorbing state)**.** *A state i is said to be* absorbing *if, when i is reached, it is impossible to leave it. The transition probability p_{ii} of an absorbing state i is: $p_{ii} = 1$.*

A DTMC is also said to be absorbing if one of its state is absorbing.

Probabilistic chain

By considering that initially a single state is occupied, one can introduce a characterization of DTMCs as probabilistic chains as defined in [Her02]. Probabilistic chains provide a simple graphical representation of DTMCs.

Definition 2.11 (Probabilistic chain)**.** *A probabilistic chain is a tuple $\langle S, \rightsquigarrow, s_0\rangle$ where:*

– *S is a nonempty set of states*

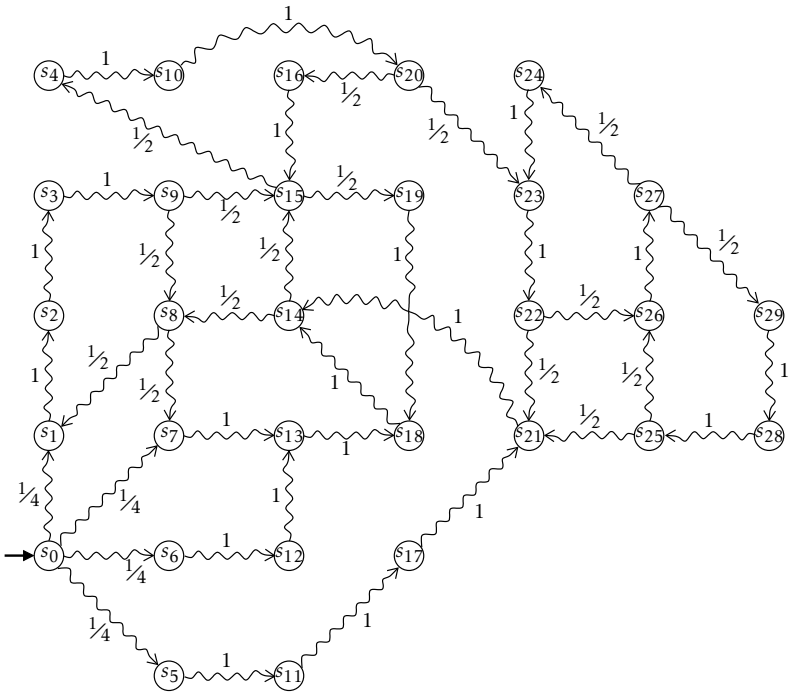


Figure 2.2: Probabilistic chain of a 1-place FIFO queue DTMC model

- $\rightsquigarrow \subset S \times]0,1] \times S \to \mathbb{N}$ *is a multi-set of probabilistic transitions such that for every state, probabilities of outgoing transitions sum up to 1, i.e.,* $\left(\forall s \in S\right), \sum_{s' \in S} \left\{\!\!\left\{ p \mid s \overset{p}{\rightsquigarrow} s' \right\}\!\!\right\} = 1$
- s_0 *is the initial state*

For convenience, we write $s \overset{p}{\rightsquigarrow} s'$ rather than $(s,p,s') \in \rightsquigarrow$. To denote that there exists $p > 0$ such that $s \overset{p}{\rightsquigarrow} s'$, we write $s \rightsquigarrow s'$.

Example 2.2. *The DTMC X modeling the FIFO queue of example 2.1 can be represented by the probabilistic chain $\mathcal{M}_P = \langle S, \rightsquigarrow, s_0 \rangle$, depicted in figure 2.2. The initial state s_0 is highlighted by an incoming straight arrow.*

A DTMC can be characterized by its transition matrix and its initial probability vector (equal to one for one state and to zero for the others), or by a probabilistic chain. Consider a DTMC $X = \{X_n \mid n \in \mathbb{N}\}$, over the state space $S = \{0,1,\ldots\}$. The transition matrix of X is $P = \left[p_{ij}\right]_{(i,j)\in S^2}$ and its initial probability vector is $\pi(0) = [\pi_i(0)]_{i \in S}$, with $\pi(0)$ satisfying $\left(\exists! k \in S\right), \pi_k(0) = 1$. The DTMC X can be characterized by the probabilistic chain $\mathcal{M}_P = \langle S, \rightsquigarrow, k \rangle$ satisfying:

- For each positive transition probability of X, there is a probabilistic transition in the probabilistic chain $\mathcal{M}_P$, i.e., $\left(\forall (i,j) \in S^2\right), \quad \left(p_{ij} = p > 0\right) \Longrightarrow i \overset{p}{\rightsquigarrow} j$
- The initial state k is the single state satisfying $\pi_k(0) = 1$

Conversely, consider a probabilistic chain $\mathcal{M}_P = \langle S, \rightsquigarrow, s_0 \rangle$ defined over the state space $S = \{0,1,\ldots\}$. $\mathcal{M}_P$ characterizes a DTMC $X = \{X_n \mid n \in \mathbb{N}\}$ over the state space $S = \{0,1,\ldots\}$. The transition matrix of X, $P = \left[p_{ij}\right]_{(i,j)\in S^2}$, and the initial probability vector of X, $\pi(0) = [\pi_i(0)]_{i \in S}$,

satisfy that:

- The transition probability p_{ij} of X, between two states i and j of S, is obtained by summing probabilities over all the possible transitions $i \rightsquigarrow j$ in $\mathcal{M}_P$, i.e., $\left(\forall (i,j) \in S^2\right), \quad p_{ij} = \sum \left\{\!\!\left\{ p \mid i \overset{p}{\rightsquigarrow} j \right\}\!\!\right\}$
- The initial probability vector is null for every state but s_0, the initial state of $\mathcal{M}_P$, for which $\pi_{s_0}(0)$ is equal to 1. In other words, we have $\pi_{s_0}(0) = 1$ and $\left(\forall j \in S, j \neq s_0\right), \pi_j(0) = 0$

Using probabilistic transitions, every time-step transition probability of a DTMC can be expressed. In a probabilistic chain, there is a probabilistic transition between two states only if the corresponding transition probability in the DTMC is non null. Notice that using multisets of probabilistic transitions in the definition of probabilistic chains enables there to be several probabilistic transitions between two states, even with the same probability.

Example 2.3. *The DTMC X modeling the FIFO queue of example 2.1 can also be represented by another probabilistic chain, different from $\mathcal{M}_P$ (depicted in figure 2.2). For instance, the transition $s_1 \overset{1}{\rightsquigarrow} s_2$ of $\mathcal{M}_P$ could be replaced by two identical transitions $s_1 \overset{1/2}{\rightsquigarrow} s_2$, whilst characterizing the same DTMC.*

Because we are targeting performance results over a long run, we are not interested in the transient behavior of an initialization phase in a DTMC. In other words, we want to take out behaviors linked to transient states since their occurrence is negligible on the long run. For the study of long-run behaviors, it is sufficient to consider a fully connected subset of the state space, i.e., a subset of recurrent states. Such a subset ensures that every state of the subset is reachable from every other state of the subset by a sequence of transitions. A fully connected DTMC is called an *irreducible* DTMC.

Definition 2.12 (Irreducible DTMC)**.** *A DTMC X defined over the state space $S \subseteq \mathbb{N}$ is said to be irreducible if:*

$$\left(\forall i, i' \in S\right) \left(\exists (j_1, \ldots, j_n) \in S^n\right) \quad p_{ij_1} > 0 \wedge p_{j_n i'} > 0 \wedge \left(\forall k \in [1,n]\right) \quad p_{j_k j_{k+1}} > 0$$

We can transpose the definition of irreducibility of a DTMC to a probabilistic chain.

Definition 2.13 (Irreducible probabilistic chain)**.** *A probabilistic chain $\mathcal{M}_P = \langle S, \rightsquigarrow, s \rangle$ is said to be irreducible if:*

$$\left(\forall i, i' \in S\right), \left(\exists (j_1, \ldots, j_n) \in S^n\right), \quad i \rightsquigarrow j_1 \rightsquigarrow \cdots \rightsquigarrow j_n \rightsquigarrow i'$$

In the following, we limit our study of DTMCs to irreducible ones, although DTMCs are not always irreducible in practice.

Lemma 2.1. *For every DTMC X over the finite state space $S \subset \mathbb{N}$ and defined by the transition matrix $P = \left[p_{ij}\right]_{(i,j) \in S^2}$, there exists an irreducible DTMC X' over the state space $S' \subset \mathbb{N}$ and defined by the transition matrix $P' = \left[p'_{ij}\right]_{(i,j) \in S'^2}$ satisfying:*

$$S' \subseteq S \quad \textit{and} \quad \left(\forall (i,j) \in S'^2\right) p'_{ij} = p_{ij}$$

The initial state of X' is chosen arbitrarily in S'. We call such a DTMC X', an irreducible DTMC induced by X.

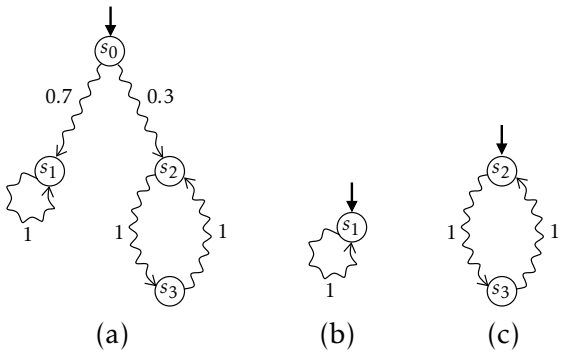


Figure 2.3: a non irreducible DTMC and two possible induced DTMCs

Proof. According to definition 2.7, we can show that if all states of a DTMC are transient, the state space of the DTMC is infinite. Because the state space is finite, there is almost one recurrent state i. The fully connected subset S_i of S including i is consequently irreducible (if i is absorbing, we have $S_i = \{i\}$). □

Several irreducible DTMCs might be induced by one that is not irreducible, and we consider in fact one of those possible induced irreducible DTMC. However, we could imagine studying all the induced DTMCs separately.

Example 2.4. *Consider the DTMC X over the state space $S = \{s_0, \ldots, s_3\}$, depicted by probabilistic chain of figure 2.3(a). There are two induced DTMCs for X: either the DTMC X' over the state space $S' = \{s_1\}$, depicted by probabilistic chain of figure 2.3(b), or the DTMC X'' over the state space $S'' = \{s_2, s_3\}$, depicted by probabilistic chain of figure 2.3(c). For X', the initial state is obviously s_1. For X'', the initial state is arbitrarily set to s_2.*

The arbitrary choice of the initial state of an induced DTMC is justified by the fact that we target long-run performance measures. Indeed, for this kind of measures, the initial state of the DTMC has no influence.

Example 2.5. *The DTMC X characterized by the probabilistic chain $\mathcal{M}_P = \langle S, \rightsquigarrow, s_0 \rangle$ of figure 2.2 is not irreducible. We can see, for instance, that the initial state s_0 cannot be reached from any of the states.*

However, there is at least an irreducible DTMC X' induced by X. For instance, we can consider the irreducible DTMC X' characterized by the probabilistic chain $\mathcal{M}'_P = \langle S', \rightsquigarrow, s_7 \rangle$, with state space $S' \subset S$, i.e., $S' = S \setminus \{s_0, s_5, s_6, s_{11}, s_{12}, s_{17}\}$. The initial state of $\mathcal{M}'_P$ is arbitrarily set to s_7. MCP' is depicted in figure 2.4. Notice that all possible induced DTMCs have the same state space S'.

2.1.3 Continuous-Time Markov Chain

Definition

In this section, we introduce notions concerning CTMCs. A full presentation of CTMCs can be found in [Nor98].

Definition 2.14 (CTMC)**.** *A stochastic process $X = \{X_t \mid t \in T\}$ is a continuous-time Markov chain if X is a continuous-time process and X is a Markov chain.*

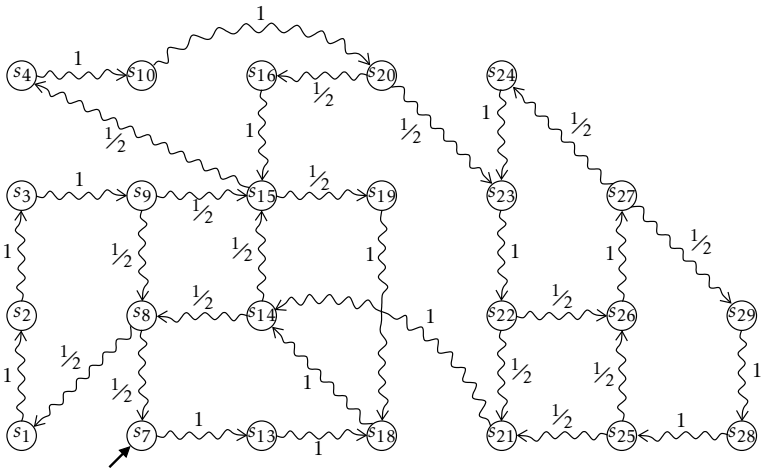


Figure 2.4: Irreducible probabilistic chain of a 1-place FIFO queue DTMC model

A CTMC moves from one state to another after a real-valued delay. Each of these moves is called a *transition*. We consider the usual time range used for continuous-time stochastic processes, i.e., $T = \mathbb{R}_+$.

Let $\{X_t \mid t \in \mathbb{R}_+\}$ be a homogeneous CTMC defined over the discrete and finite state space $S \subset \mathbb{N}$. For every pair of states $(i, j) \in S^2$, we define a *transition rate at time t*, $r_{ij}(t)$, from state i to state j,

$$r_{ij}(t) = \begin{cases} \lim_{\Delta t \to 0} \frac{\Pr\{X_{t+\Delta t}=j \mid X_t=i\}}{\Delta t} & \text{if } i \neq j, \\ 0 & \textit{otherwise.} \end{cases}$$

Because each time point is a renewal point in a homogeneous CTMC, the transition rate can be rewritten:

$$\left(\forall t \in \mathbb{R}_+\right), \quad r_{ij}(t) = r_{ij}(0) = \begin{cases} \lim_{\Delta t \to 0} \frac{\Pr\{X_{\Delta t}=j \mid X_0=i\}}{\Delta t} & \text{if } i \neq j, \\ 0 & \textit{otherwise.} \end{cases}$$

Thus, we define $r_{ij} = r_{ij}(0)$, and r_{ij} is simply called *transition rate* from state i to state j.

In a homogeneous CTMC, the sojourn time SJ_i in a state $i \in S$ is the time spent in i before moving to another state. It is characterized by the set of transition rates from i, $\left\{r_{ij}\right\}_{j \in S}$. Indeed, for every state $i \in S$, SJ_i follows an exponential distribution, $\exp(\lambda)$, parameterized by $\lambda = \sum_{j \in S} r_{ij}$. Its cumulative distribution function is defined by:

$$\Pr\{\mathsf{SJ}_i \leq t\} = 1 - \exp^{-\left(\sum_{j \in S} r_{ij}\right)t}$$

As in the discrete-time case, the exponential distribution is the only memoryless continuous probability distribution. The memoryless property of the distribution of the sojourn time in a CTMC is also a direct consequence of the Markov property.

For a given state i, the distribution of the sojourn time in i, SJ_i, is also equal to the distribution of the minimum of the distributions $\{\exp^{r_{ij}}\}_{j \in S}$ (the class of exponential distributions is closed for the minimum function). In the particular case where a single transition from state i is possible, i.e., a single state j such that $r_{ij} > 0$, the distribution of SJ_i follows the exponential distribution parameterized by r_{ij}. The transition from i to j happens consequently after an r_{ij}-exponentially distributed delay. In the general case, when several transitions from state i

are possible, there is a set $S' \in S$ such that $\left(\forall j \in S'\right), r_{ij} > 0$ (we consider S' as the largest set satisfying this property). For every state $j \in S'$, the transition from i to j potentially happens after an r_{ij}-exponentially distributed delay. Consequently, the transition from i will be the one that happens earlier, i.e., the one that realizes the minimum of all the r_{ij}-exponential distributions.

From transition rates, we construct a matrix $Q = \left[q_{ij}\right]_{(i,j)\in S^2}$, called the *transition rate matrix*, that satisfies

$$\left(\forall (i,j) \in S^2\right), \quad q_{ij} = \begin{cases} r(i,j) & \text{if } i \neq j \\ -\sum_{j' \in S} r(i,j') & \text{if } i = j \end{cases}$$

Consider the vector $\pi(t) = [\pi_i(t)]_{i\in S}$, where for every state $i \in S$, $\pi_i(t) = \mathsf{Pr}\{X_t = i\}$. $\pi(t)$ is called *probability vector* of the process at time t and corresponds to the probability distribution of the process over the state space at time t. These probability vectors satisfy $\left(\forall t \in \mathbb{R}_+\right), \sum_{i\in S} \pi_i(t) = 1$.

A homogeneous CTMC is fully characterized by its transition rate matrix and its initial probability vector $\pi(0)$. Indeed, all possible evolutions can be computed from the initial probability vector and the transition rate matrix of a CTMC. In the rest of the thesis, we limit our study to CTMCs for which a single state is initially occupied. In practice, this assumption is in most cases verified.

Example 2.6. *Consider the hardware FIFO queue presented in example 1.1 with the following parameters in a continuous-time context:*

- *The queue size Q_S is set to one place, $Q_S = 1$*
- *The hardware related delays D_{PUSH} (push operation delay) and D_{POP} (pop operation delay) are exponentially distributed with parameters λ and μ*
- *The software-related delays D_{PROD} (production delay) and D_{CONS} (consumption delay) are exponentially distributed too, with parameters δ_1 and δ_2*

Initially, we assume that the queue is empty, and that production and consumption delays have to elapse before first insertions or extractions in the FIFO queue. This system can be modeled using a CTMC X defined over a state space $S = \{s_0, \ldots, s_7\}$. The transition rate matrix $Q = \left[q_{s_i s_j}\right]_{(s_i,s_j)\in S^2}$ and the initial probability vector $\pi(0) = \left[\pi_{s_i}(0)\right]_{s_i \in S}$ of X are depicted in figure 2.5

State properties

As in the discrete-time case, states can be classified according to some properties. Consider the homogeneous CTMC $\{X_t \mid n \in \mathbb{R}_+\}$ defined over the discrete and finite state space $S \subset \mathbb{N}$, and characterized by its transition rate matrix $Q = \left[q_{ij}\right]_{(i,j)\in S^2}$ and its initial probability vector $\pi(0)$.

We define the probability ret_i of ever returning to state i when starting in i by

$$\mathrm{ret}_i = \mathsf{Pr}\{\inf(t > 0 : X_t = i \mid X_0 = i) < +\infty\}.$$

Definition 2.15 (Transient and recurrent states)**.** *A state $i \in S$ is said to be* recurrent *if, starting in i, the probability ret_i to ever return to i is 1. If, starting in i, there is a non-zero probability to never return in i ($ret_i < 1$), the state is said to be* transient.

$$Q = \begin{bmatrix} -(\delta_1+\delta_2) & \delta_1 & 0 & 0 & \delta_2 & 0 & 0 & 0 \\ 0 & -(\lambda+\delta_2) & \lambda & 0 & 0 & \delta_2 & 0 & 0 \\ 0 & 0 & -(\delta_1+\delta_2) & \delta_1 & 0 & 0 & \delta_2 & 0 \\ 0 & 0 & 0 & -\delta_2 & 0 & 0 & 0 & \delta_2 \\ 0 & 0 & 0 & 0 & -\delta_1 & \delta_1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -\lambda & \lambda & 0 \\ \mu & 0 & 0 & 0 & 0 & 0 & -(\delta_1+\mu) & \delta_1 \\ 0 & \mu & 0 & 0 & 0 & 0 & 0 & -\mu \end{bmatrix}$$

$$\pi(0) = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Figure 2.5: Transition rate matrix and initial probability vector of a 1-place FIFO queue CTMC model

Then, we define the mean recurrence time mean_i of state i:

$$\mathsf{mean}_i = \lim_{t\to+\infty} \frac{1}{t} \int_0^t \inf(t > 0 : X_t = i \mid X_0 = i)\, dt,$$

i.e., the average time needed, starting in i, to return to i.

Definition 2.16 (positive recurrent state)**.** *A state $i \in S$ is said to be* positive recurrent *if, starting in i, the mean recurrence time mean_i is finite, i.e., $\mathsf{mean}_i < +\infty$.*

Definition 2.17 (ergodic state)**.** *a state $i \in S$ is said to be* ergodic, *if it is positive recurrent.*

A CTMC is also said to be ergodic if all its states are ergodic.

Definition 2.18 (absorbing state)**.** *A state i is said to be* absorbing *if, when i is reached, it is impossible to leave it. The factor q_{ii} of the transition rate matrix of an absorbing state i satisfies: $q_{ii} = 0$ (which implies that q_{ij} is null for all j).*

A CTMC is also said to be absorbing if one of its state is absorbing.

Markovian chain

By considering that initially a single state is occupied, one can introduce a characterization of CTMCs as Markovian chains as defined in [Her02]. Markovian chains provide a simple graphical representation of CTMCs.

Definition 2.19 (Markovian chain)**.** *A Markovian chain is a tuple $\langle S, \dashrightarrow, s_0 \rangle$ where:*

- *S is a nonempty set of states*
- *$\dashrightarrow \subset S \times \mathbb{R}_+ \times S \to \mathbb{N}$ is a multi-set of Markovian transitions, which are transitions with transition rates*
- *s_0 is the initial state*

For convenience, we write $s \overset{\lambda}{\dashrightarrow} s'$ rather than $(s, \lambda, s') \in \dashrightarrow$. To denote that there exists $\lambda > 0$ such that $i \overset{\lambda}{\dashrightarrow} j$, we write $s \dashrightarrow s'$.

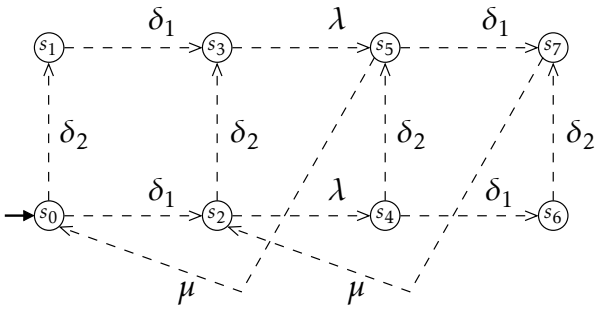


Figure 2.6: Markovian chain of a 1-place FIFO queue CTMC model

Example 2.7. *The CTMC X modeling the FIFO queue of example 2.6 can be represented by the Markovian chain $\mathcal{M}_M = \langle S, \dashrightarrow, s_0 \rangle$, depicted in figure 2.6. The initial state s_0 is highlighted by an incoming straight arrow.*

A CTMC can be characterized by its transition matrix and initial probability vector (equal to one for a single state), or by a Markovian chain. Consider a CTMC $X = \{X_t \mid t \in \mathbb{R}_+\}$, over the state space $S = \{0, 1, \ldots\}$. The transition rate matrix of X is $Q = \left[q_{ij}\right]_{(i,j)\in S^2}$ and its initial probability vector is $\pi(0) = [\pi_i(0)]_{i\in S}$, with $\pi(0)$ satisfying $\left(\exists! k \in S\right), \pi_k(0) = 1$. The CTMC X can be characterized by the Markovian chain $\mathcal{M}_M = \langle S, \dashrightarrow, s \rangle$ satisfying:

- For each positive transition rate of X, there is a Markovian transition in the Markovian chain $\mathcal{M}_M$, i.e., $\left(\forall (i,j) \in S^2\right), \quad q_{ij} > 0 \Longrightarrow i \overset{q_{ij}}{\dashrightarrow} j$
- The initial state s is equal to the single state k satisfying $\pi_k(0) = 1$

Conversely, consider a Markovian chain $\mathcal{M}_M = \langle S, \dashrightarrow, s \rangle$ defined over the state space $S = \{0, 1, \ldots\}$. $\mathcal{M}_M$ characterized a CTMC $X = \{X_t \mid t \in \mathbb{R}_+\}$ over the state space S. The transition rate matrix of X, $Q = \left[q_{ij}\right]_{(i,j)\in S^2}$, and the initial probability vector of X, $\pi(0) = [\pi_i(0)]_{i\in S}$, satisfy:

- The transition rate r_{ij} between two different states i and j of S is obtained by summing rates over all the possible transitions $i \dashrightarrow j$ in $\mathcal{M}_M$, i.e., $\left(\forall (i,j) \in S^2,\ i \neq j\right), \quad q_{ij} = \sum \left\{\!\!\left\{ \lambda \mid i \overset{\lambda}{\rightsquigarrow} j \right\}\!\!\right\}$. When states i and j are identical, we have $q_{ii} = -\sum_{j\in S, j\neq i} q_{ij}$.
- The initial probability vector is null for every state but s, the initial state of $\mathcal{M}_M$, for which $\pi_s(0)$ is equal to 1. In other words, we have $\pi_s(0) = 1$ and $\left(\forall j \in S,\ j \neq s\right), \quad \pi_j(0) = 0$.

Using Markovian transitions, every transition rate of a CTMC can be expressed. In a Markovian chain, there is a Markovian transition between two states only if the corresponding transition rate in the CTMC is non null. Notice that using multisets of Markovian transitions in the definition of Markovian chains permits there to be several Markovian transitions between two states, even with the same transition rate.

Example 2.8. *The CTMC X modeling the FIFO queue of example 2.6 can be represented by another Markovian chain, different from $\mathcal{M}_M$ (depicted in figure 2.6. For instance, the transition $s_0 \overset{\delta_2}{\dashrightarrow} s_1$ of $\mathcal{M}_M$ can be replaced by two identical Markovian transitions $s_0 \overset{\delta_2/2}{\dashrightarrow} s_1$, whilst characterizing the same CTMC.*

Because we are targeting performance results for a long run, we are not interested in the transient behavior of an initialization phase in a CTMC. In other words, we want to take out

behaviors linked to transient state as in the discrete-time case. For the study of long-run behaviors, it is sufficient to consider a fully connected subset of the state space, i.e., a subset of recurrent states. Such a subset ensures that every state of the subset is reachable from every other state of the subset by a sequence of transitions. A fully connected CTMC is called an *irreducible* CTMC.

Definition 2.20 (Irreducible CTMC)**.** *A CTMC X defined over the state space $S \subseteq \mathbb{N}$ is said to be irreducible if:*

$$\left(\forall i, i' \in S\right)\left(\exists (j_1, \ldots, j_n) \in S^n\right) \quad r_{ij_1} > 0 \wedge r_{j_n i'} > 0 \wedge \left(\forall k \in [1, n]\right) \quad r_{j_k j_{k+1}} > 0$$

We can transpose the definition of irreducibility of a CTMC to a Markovian chain.

Definition 2.21 (Irreducible Markovian chain)**.** *A Markovian chain $\mathcal{M}_M = \langle S, \dashrightarrow, s\rangle$ is said to be irreducible if:*

$$\left(\forall i, i' \in S\right), \left(\exists (j_1, \ldots, j_n) \in S^n\right), \quad i \dashrightarrow j_1 \dashrightarrow \cdots \dashrightarrow j_n \dashrightarrow i'$$

In the following, we limit our study of CTMCs to irreducible ones, although CTMCs are not always irreducible in practice.

Lemma 2.2. *For every CTMC X over the finite state space $S \subset \mathbb{N}$ and defined by the transition rate matrix $Q = \left[q_{ij}\right]_{(i,j)\in S^2}$, there exists an irreducible CTMC X' over the state space $S' \subset \mathbb{N}$ and defined by the transition rate matrix $Q' = \left[q'_{ij}\right]_{(i,j)\in S^2}$ satisfying:*

$$S' \subseteq S \quad \textit{and} \quad q'_{ij} = q_{ij}$$

The initial state of X' is chosen arbitrarily in S'. We call such a CTMC X', an irreducible CTMC induced by X.

Proof. The proof in the continuous-time case follows the same lines than the proof in the discrete-time case (cf. lemma 2.1). □

Several irreducible CTMCs might be induced by one that is not irreducible, and we consider, as for DTMCs, one of those possible induced irreducible CTMCs. The arbitrary choice of the initial state of an induced CTMC is also justified by the fact that we target long-run performance measures.

Example 2.9. *The CTMC X characterized by the Markovian chain depicted in figure 2.6 is irreducible.*

2.1.4 Steady-State and Transient Probabilities

In most cases, analysis of Markov chains relies on the computation of state probabilities. In other words, for a Markov chain $\{X_t\}$ (in discrete or continuous time setting) defined over a state space S, the study of the probability vector $\pi(t) = [\pi_i(t)]_{i\in S}$ is of main interest. $\pi(t)$ is generally called *transient probability vector* (because it depends on time).

We are particularly interested in the limit behavior of the transient probability vector when time tends to infinity ($\lim_{t\to\infty} \pi(t)$). This limit exists only if an equilibrium is reached by the Markov chain. When it exists, this limit is called *steady-state probability vector* and is written $\pi = [\pi_i]_{i\in S}$.

State Probabilities of Discrete-Time Markov Chains

Consider a homogeneous DTMC $X = \{X_n \mid n \in \mathbb{N}\}$, over the state space $S = \{0, 1, \ldots\}$, characterized by the transition matrix $P = \left[p_{ij}\right]_{(i,j)\in S^2}$ and the initial probability vector $\pi(0) = [\pi_i(0)]_{i\in S}$, satisfying $\left(\exists! k \in S\right), \pi_k(0) = 1$.

Due to homogeneity, the probability vector at time $n+1$, $\pi(n+1)$, is given by the product of the probability vector at time n, $\pi(n)$, by the transition matrix P, i.e.,

$$\left(\forall n \in \mathbb{N}\right), \quad \pi(n+1) = \pi(n) \times P.$$

We can easily obtain a value of the probability vector at time n as a function of the initial probability vector: the probability vector of a DTMC at time n is obtained by multiplying the initial probability vector with the transition matrix to the power of n.

Definition 2.22 (Computation of transient probability vectors)**.** *Consider a homogeneous DTMC $X = \{X_n \mid n \in \mathbb{N}\}$ characterized by the transition matrix P and the initial probability vector $\pi(0)$. The transient probability vector at time n, $\pi(n)$, is given by:*

$$\left(\forall n \in \mathbb{N}\right), \quad \pi(n) = \pi(0) \times P^n$$

To compute the steady-state probability vector π, we have to ensure the convergence of the transient probability vector when time n tends to infinity. For a DTMC, the steady-state probability vector exists if the DTMC is ergodic, which is also simply ensured if the DTMC is irreducible and aperiodic.

The ergodicity property of a DTMC enables us to compute its steady-state probability vector π as the limit of $\pi(n)$ with n tending to infinity. It satisfies the fixed point equation: $\pi = \pi \times P$. The computation of the steady-state probability vector is essentially the resolution of the a linear system $\pi \times (P - I) = 0$, where I is the identity matrix, and 0 the null vector. This system does not necessarily have a single solution. The only solution we are interested in is a probabilistic distribution over the state space. In other words, we require that the solution satisfies: $\sum_{i\in S} \pi_i = 1$. Notice that the steady state probability vector is independent from the initial probability vector.

Definition 2.23 (Steady-state probability vector calculation)**.** *Consider an ergodic homogeneous DTMC X characterized by the transition matrix P. The steady state probability vector, π, satisfies:*

$$\begin{cases} \pi \times (P - I) = 0 \\ \sum_{i\in S} \pi_i = 1 \end{cases}$$

Unfortunately, computation of the steady-state probability vector of a DTMC force us to assume the aperiodicity property, in addition to the irreducible property (still assumed as prerequisite for the DTMC being studied). We prefer not to limit our study to aperiodic DTMCs.

Example 2.10. *The DTMC characterized by the probabilistic chain depicted in figure 2.7 is periodic. Indeed, the number of transitions needed to return in each state s_1 and s_2 is a multiple of two. Its transient state probabilities do not converge. Indeed,*

$$\pi_{s_0}(n) = \begin{cases} 1 & \text{if } n \text{ is even} \\ 0 & \text{otherwise} \end{cases} \qquad \text{and} \quad \pi_{s_1}(n) = \begin{cases} 0 & \text{if } n \text{ is even} \\ 1 & \text{otherwise} \end{cases}$$

The transient probability of s_0 or s_1 alternates between 0 and 1 in time without converging.

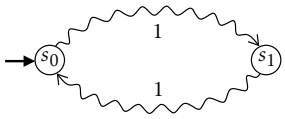


Figure 2.7: A simple periodic probabilistic chain

To avoid dealing with periodic considerations, we consider the long-run average vector $\widetilde{\pi} = [\widetilde{\pi}_i]_{i \in S}$ of transient probabilities using the Cesáro limit construction.

Definition 2.24 (Cesáro limit)**.** *Given a sequence $a = (a_i)_{i \in \mathbb{N}}$, the sequence of Cesáro averages $c = (c_n)_{n \in \mathbb{N}}$ is defined by:*

$$c_n = \frac{1}{n} \sum_{k=1}^{n} a_k$$

If it exists, the Cesáro limit of the sequence a is the limit to infinity of the sequence of its Cesáro averages c_n.

For a time-homogeneous irreducible DTMC, the long-run average vector $\widetilde{\pi}$, is ensured to exist [Tij03] and is constructed as the Cesáro limit of the sequence of transient probability vectors, $(\pi(n))_{n \in \mathbb{N}}$.

Definition 2.25 (DTMC long-run average vector)**.** *The long-run average vector $\widetilde{\pi} = [\widetilde{\pi}_i]_{i \in S}$ of a DTMC is given by:*

$$\Big(\forall i \in S\Big), \quad \widetilde{\pi}_i = \lim_{n \to \infty} \left(\frac{1}{n} \sum_{k=0}^{n} \pi_i(k) \right)$$

When $\pi(n)$ converges, its Cesáro limit $\widetilde{\pi}$ is known to coincide with the limit of $\pi(n)$ (i.e., the steady state probability π). Informally, the long-run average vector corresponds to the average fraction of time spent in each state. In the rest of the thesis, we are mainly interested in the average fraction of time spent in each state. Consequently, when talking about the steady-state probabilities of a DTMC, we refer to long-run averages and we will write π instead of $\widetilde{\pi}$.

Considering long-run average vectors instead of steady-state probability vectors allows us not to limit the study to ergodic DTMCs, but to all irreducible ones. It implies that the results presented later can be theoretically applied to arbitrary irreducible DTMCs considering the long-run averages. In practice, an aperiodic DTMC is preferable because of facilities concerning the computation of steady state probabilities.

Example 2.11. *Because the DTMC depicted in figure 2.7 is periodic, the transient state probabilities do not converge. However, the Cesáro limit of the sequences $\left(\pi_{s_0}(n)\right)_{n \in \mathbb{N}}$ and $\left(\pi_{s_1}(n)\right)_{n \in \mathbb{N}}$ of transient probabilities of s_0 and s_1 exists. Using $[r]$ as notation for the floor function applied to an arbitrary real number r, the n-th Cesáro averages of the transient probability sequences are given by:*

$$\widetilde{\pi}_{s_0}(n) = \tfrac{1}{n} \textstyle\sum_{i=0}^{n} \pi_{s_0}(i) = \frac{\left[\frac{n}{2}\right]}{n} \quad \xrightarrow[n \to \infty]{} \quad \tfrac{1}{2}$$
$$\widetilde{\pi}_{s_1}(n) = \tfrac{1}{n} \textstyle\sum_{i=0}^{n} \pi_{s_1}(i) = \frac{\left[\frac{n+1}{2}\right]}{n} \quad \xrightarrow[n \to \infty]{} \quad \tfrac{1}{2}$$

The Cesáro averages sequences converge to $\frac{1}{2}$ that is the long run average fraction of time spent in s_0 and s_1, $\widetilde{\pi}_{s_0} = \widetilde{\pi}_{s_1} = \frac{1}{2}$.

State Probabilities of Continuous-Time Markov Chains

Consider a homogeneous CTMC $X = \{X_t \mid t \in \mathbb{R}_+\}$, over the state space $S = \{0, 1, \ldots\}$, characterized by the transition rate matrix $Q = \left[q_{ij}\right]_{(i,j)\in S^2}$ and the initial probability vector $\pi(0) = [\pi_i(0)]_{i\in S}$, satisfying $\left(\exists! k \in S\right), \pi_k(0) = 1$. The transient probability vector of a CTMC satisfies a differential equation function of the transition rate matrix Q.

Definition 2.26 (Computation of transient probability vectors)**.** *Consider a homogeneous CTMC $\{X_t \mid t \in \mathbb{R}_+\}$ characterized by the transition rate matrix Q and the initial probability vector $\pi(0)$. The transient probability vector at time t, $\pi(t)$, satisfies the differential equation:*

$$\frac{\mathrm{d}\pi(t)}{\mathrm{d}t} = \pi(t) \times Q \quad \textit{with initial condition } \pi(0) \tag{1}$$

As for DTMCs, to compute the steady-state probability vector π, we have to ensure the convergence of the transient probability vector when time t tends to infinity. This convergence is ensured when a CTMC is ergodic, i.e., when it is irreducible.

We assumed previously that CTMCs studied are irreducible, thus ergodic. This assumption ensures the existence of the steady-state probability vector. For the steady-state probability vector, the equation (1) is satisfied too. Due to convergence we have time independence, and the equation can be rewritten as:

$$0 = \pi \times Q$$

Hence, the computation of the steady-state probability vector boils down to the resolution of a linear system of equations. As for DTMCs, the solution we are interested in is a probability distribution over the state space satisfying this equation.

2.2 Annotated Markov Chains

2.2.1 Definitions

Markov chains are the starting point for performance evaluation. However, the computation of many complex measures requires extra-information, not present in a Markov chain. Usually, a reward function $\mathfrak{K} : S \mapsto \mathbb{R}$, defined over the state space S of the Markov chain, is used to associate a real value to each state [Her02]. We generalize this approach by associating to each state an annotation that is not necessarily a real value. To this aim, we define an annotation function over the state space S of a Markov chain.

Definition 2.27 (Annotation function)**.** *Consider a set A of arbitrary objects, endowed with the equality ($=$) and inequality ($\neq$) comparison operators. An Annotation function $\mathscr{A} : S \mapsto A$ over a state space S of a Markov chain is a function that associates to each state $s \in S$ an element of the set A.*

Using an annotation function, annotations are integrated directly in a more general formalism than Markov chains: the *annotated Markov chains* (AMCs). We define annotated Markov chains using their probabilistic chain or Markovian chain characterization.

Definition 2.28 (DTAMC)**.** *A Discrete-Time Annotated Markov Chain is a tuple $\langle S, \rightsquigarrow, s, \mathscr{A}\rangle$, where:*

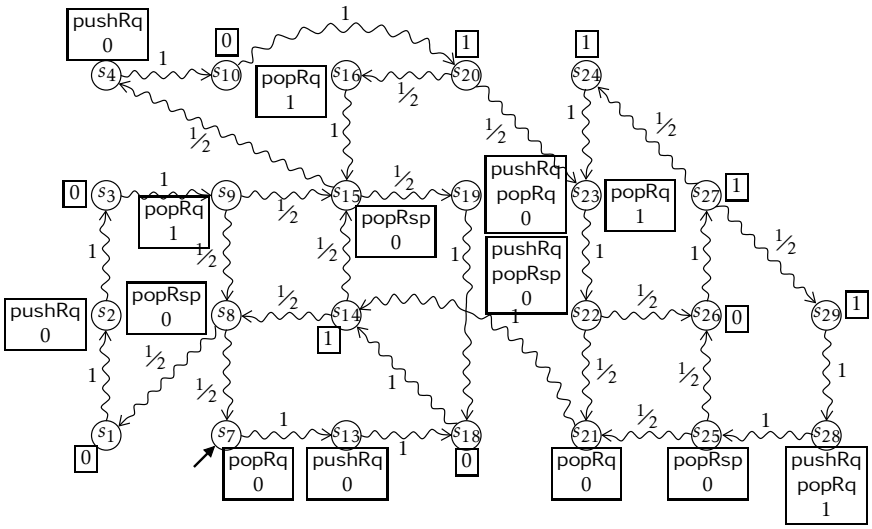


Figure 2.8: DTAMC corresponding to a 1-place FIFO queue DTMC model

- $\langle S, \rightsquigarrow, s\rangle$ *is a probabilistic chain*
- $\mathscr{A}$ *is an annotation function defined over* S

Definition 2.29 (CTAMC)**.** *A Continuous-Time Annotated Markov Chain is a tuple* $\langle S, \dashrightarrow, s, \mathscr{A}\rangle$*, where:*

- $\langle S, \dashrightarrow, s\rangle$ *is a Markovian chain*
- $\mathscr{A}$ *is an annotation function defined over* S

We can easily show that AMCs are more expressive than MCs. Indeed, every MC can be expressed as an AMC. Consider a set $A = \{\varnothing\}$ containing one single element $\varnothing$. For a Markov chain $\mathcal{M}$ over a state space S, define the annotation function $\mathscr{A} : S \mapsto A$ such that $\left(\forall s \in S\right), \mathscr{A}(s) = \varnothing$. The Markov chain $\mathcal{M}$ can be expressed as an AMC, considering the annotation function $\mathscr{A}$. Moreover, all states having the same annotation, there is no more information in the constructed AMC than in the MC.

We illustrate the use of AMCs in the two following examples: the first one in discrete-time setting and the second one in continuous-time setting.

Example 2.12. *Consider the DTMC modeling the FIFO queue of example 2.1 and characterized by the irreducible probabilistic chain depicted in figure 2.4. Call* $\mathcal{M}_P = \langle S, \rightsquigarrow, s_7\rangle$ *this DTMC. We construct a DTAMC* $\mathcal{M}_P^{\mathscr{A}} = \langle S, \rightsquigarrow, s_7, \mathscr{A}\rangle$ *over* $\mathcal{M}_P$ *with the annotation function* $\mathscr{A} : S \mapsto \{\mathsf{True}, \mathsf{False}\}^3 \times \mathbb{N}$ *that associates to each state three booleans values* pushRq*,* popRq*,* popRsp*, and an integer value* size*.* pushRq *is* True *if the initiation of an element insertion in the queue (push request) occurs.* popRq *is* True *if the initiation of an element extraction from the queue (pop request) occurs.* popRsp *is* True*, if the end of an element extraction from the queue (pop response) occurs.* size *denotes the current size of the queue in the state. The DTAMC* $\mathcal{M}_P^{\mathscr{A}}$ *is depicted in figure 2.8. On this figure, each state is labeled with the current size of the queue and* pushRq*,* popRq *or* popRsp *if those booleans are* True*.*

Example 2.13. *Consider the CTMC modeling the FIFO queue of example 2.6 and depicted by the Markovian chain of figure 2.6. Consider the annotation function* $\mathscr{A} : S \mapsto \mathbb{N}$ *that associates to each*

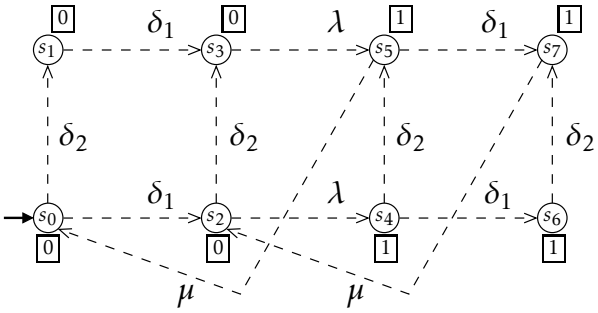


Figure 2.9: CTAMC corresponding to a 1-place FIFO queue CTMC model

state the current size of the queue. The corresponding CTAMC is depicted in figure 2.9. Each state is labeled with the current size of the queue.

2.2.2 Performance Measures on Annotated Markov Chains

As underlined previously, state probabilities are only the starting point for performance evaluation. Indeed, the information contained in state probabilities, despite its importance, is difficult to exploit on large Markov chains. For instance, considering a one million state Markov chain, what is the steady-state probability of a particular state? This information is useful only if we can associate this state to a particular functional behavior of the modeled system. Moreover a single functional behavior of the modeled system may correspond to several states of the Markov chain model.

Example 2.14. *Consider the CTMC model of the one-place queue of example 2.6 and depicted by the Markovian chain of figure 2.6. For instance we are interested in the functional behavior of the queue "The queue is empty". An interesting measure is the average fraction of time the queue is empty. Just as with the CTMC model, it is not possible to answer this question. Using information provided by the CTAMC depicting the same example (figure 2.9), we can say that the queue is empty in states s_0, s_1, s_2 and s_3 of the CTAMC model. This information allows us to answer the question: the average fraction of time the queue is empty corresponds to the sum of steady-state probabilities of the states s_0 to s_3.*

Annotated Markov chains provide enough information to compute a wide class of performance results. Indeed, we can associate any kind of functional information to states of a Markov chain. We generalize the computation of complex performance results using reward functions to the computation of complex performance results using AMCs. We define our own reward function computed from annotations of an AMC (instead of defining rewards directly on states). The advantage of the use of AMCs is that annotations can be used to group states as illustrated in example 2.14.

Definition 2.30 (Reward function)**.** *Consider an AMC, defined over a state space S, and endowed with the annotation function $\mathscr{A} : S \mapsto A$. A reward function, $\mathfrak{R}$ is a function used to get a real value from annotations associated with each state, i.e., $\mathfrak{R} : A \mapsto \mathbb{R}$.*

A wide class of performance measures is then given by sum of states probabilities, weighted by rewards associated to states, as underlined in [Her02]. We provide a definition of this kind of measures for transient and for steady-state probabilities.

Definition 2.31 (Time-dependent performance measures)**.** *Consider an AMC defined over a state space $S = \{s_1, \ldots, s_n\}$, having the annotation function $\mathscr{A} : S \mapsto A$ and a reward function $\mathfrak{R} : A \mapsto \mathbb{R}$. For each time instant t (either in continuous-time settings with a CTAMC or in discrete-time settings with a DTAMC), the expression:*

$$\sum_{s_i \in S} \pi_{s_i}(t) \mathfrak{R}(\mathscr{A}(s_i))$$

defines a performance measure on the AMC at time t.

Definition 2.32 (Long-run performance measures)**.** *Consider an AMC defined over a state space $S = \{s_1, \ldots, s_n\}$, and having the annotation function $\mathscr{A} : S \mapsto A$, and a reward function $\mathfrak{R} : A \mapsto \mathbb{R}$. The expression:*

$$\sum_{s_i \in S} \pi_{s_i} \mathfrak{R}(\mathscr{A}(s_i))$$

defines a performance measure on the AMC on the long-run.

Example 2.15. *We can formalize the result given in example 2.14 using a reward function. The annotation function of the CTAMC depicted in figure 2.9 is $\mathscr{A} : S \mapsto \mathbb{N}$. Let us consider the reward function $\mathfrak{R} : \mathbb{N} \mapsto \{0,1\}$ such that $\mathfrak{R}(n) = 1$ if $n = 0$, and $\mathfrak{R}(n) = 0$ otherwise. In other words, the reward associated with each state is 1 only if the current size of the queue is 0. The average fraction of time the queue is empty is given by:*

$$\sum_{s_i \in \{s_1, \ldots, s_7\}} \pi_{s_i} \mathfrak{R}(\mathscr{A}(s_i))$$

2.2.3 Equivalences on Annotated Markov Chains

Equivalence relations for comparing Markov chains are important in practice. Moreover they can be used to aggregate states of a Markov chain, finding the smallest equivalent Markov chain. These aggregations have to preserve properties concerning the system behavior. The notion of aggregation generally known as lumpability [Hil96] relies on the definition of a suitable partitioning of the state space, which is ensured to remain a Markov chain. We prefer using the definition of bisimulations introduced in [Her02] for probabilistic and Markovian chains, because those bisimulations match with general definitions of lumpability avoiding the need to define a suitable partition on state space. Moreover, bisimulations ensure that the aggregation still leads to a probabilistic or Markovian chain. We directly present those bisimulations on annotated state Markov chains, adapting the reward preserving bisimulations presented in [Her02] to AMCs.

Bisimulation of Discrete-Time Annotated Markov Chains

For every DTAMC $\mathcal{M}_P = \langle S, \rightsquigarrow, s, \mathscr{A} \rangle$, consider the cumulative probability function $\gamma_P : S \times 2^S \mapsto [0,1]$ that cumulates the transition probabilities from a state $s' \in S$ to a set of states $\mathscr{S} \subseteq S$:

$$\gamma_P(s', \mathscr{S}) = \sum_{s'' \in \mathscr{S}} \left\{\!\!\left\{ p \mid s' \overset{p}{\rightsquigarrow} s'' \right\}\!\!\right\}$$

Definition 2.33 (DTAMC probabilistic bisimulation)**.** *Given a DTAMC* $\mathcal{M}_P = \langle S, \rightsquigarrow, s, \mathscr{A} \rangle$*, probabilistic bisimulation equivalence* ($\sim_p$) *is the coarsest equivalence relation on* $\mathcal{M}_P$ *such that* $\left(\forall (s_1, s_2) \in S^2\right)$*,* $s_1 \sim_p s_2$ *implies that for each equivalence class* $\mathcal{C}$ *of* $\sim_p$*,*

$$\begin{aligned} \gamma_P(s_1, \mathcal{C}) &= \gamma_P(s_2, \mathcal{C}) \\ \mathscr{A}(s_1) &= \mathscr{A}(s_2) \end{aligned}$$

Two DTAMCs are probabilistically bisimilar if their initial states are equivalent according to the DTAMC probabilistic bisimulation ($\sim_p$).

Note that this bisimulation can be applied to a pure probabilistic chain, considering it as a DTAMC, as seen in section 2.2.1 (each state receives the same annotation). On a pure probabilistic chain, this bisimulation has an interesting property: the second condition (equality of annotations) is always true. Only the first equation remains. But this equation implies that all the states are equivalent. Indeed, for each state $s' \in S$, outgoing transitions sum to 1 by definition, which implies $\gamma_P(s', S) = 1$. It means that the probabilistic bisimulation is not interesting for pure probabilistic chains: all probabilistic chains are equivalent.

In addition to comparing DTAMCs, the probabilistic bisimulation $\sim_p$ can be used to get the smallest DTAMC (in terms of state space size) equivalent to another one. Getting the smallest equivalent DTAMC consists in minimizing the state space of the model. This smallest equivalent DTAMC is called the *quotient*.

Definition 2.34 (DTAMC quotient)**.** *Given a DTAMC* $\mathcal{M}_P = \langle S, \rightsquigarrow, s, \mathscr{A} \rangle$*, the quotient of* $\mathcal{M}_P$ *is the DTAMC* $\mathcal{M}_{P/_{\sim_p}}$ *defined over the state space* $S_{/_{\sim_p}}$ *satisfying:*

- $\mathcal{M}_{P/_{\sim_p}}$ *and* $\mathcal{M}_P$ *are equivalent* ($\mathcal{M}_{P/_{\sim_p}} \sim_p \mathcal{M}_P$)
- *for every DTAMC* $\mathcal{M}'_P$*, defined over the state space* S'*, and equivalent to* $\mathcal{M}_P$ *(and to* $\mathcal{M}_{P/_{\sim_p}}$*), we have* $\mathsf{card}(S_{/_{\sim_p}}) \leq \mathsf{card}(S')$

We say that a DTAMC is minimal if its size is the same as the size of its quotient.

Example 2.16. *The DTAMC* $\mathcal{M}_P^{\mathscr{A}} = \langle S, \rightsquigarrow, s_7, \mathscr{A} \rangle$ *depicted in figure 2.8 is minimal.*

Now consider the DTAMC $\mathcal{M}_P^{\mathscr{A}'} = \langle S, \rightsquigarrow, s_7, \mathscr{A}' \rangle$ *with an annotation function* $\mathscr{A}'$ *similar to* $\mathscr{A}$*, but limited to booleans* popRq *and* popRsp *(*$\mathscr{A}' : S \mapsto \{\mathsf{True}, \mathsf{False}\}^2$*).* $\mathcal{M}_P^{\mathscr{A}}$ *and* $\mathcal{M}_P^{\mathscr{A}'}$ *only differ in their annotation functions.*

A representation of the DTAMC $\mathcal{M}_P^{\mathscr{A}'}$ *could be given by figure 2.8, ignoring size and* pushRq *values in annotations.* $\mathcal{M}_P^{\mathscr{A}'}$ *is not minimal any more. Sets* $\{s_{24}, s_{29}\}$*,* $\{s_{23}, s_{28}\}$ *and* $\{s_{25}, s_{22}\}$ *are equivalent classes with respect to* $\sim_p$*. The quotient* $\mathcal{M}_{P/_{\sim_p}}^{\mathscr{A}'}$ *of* $\mathcal{M}_P^{\mathscr{A}'}$ *is depicted in figure 2.10.*

Bisimulation of Continuous-Time Annotated Markov Chains

For every CTAMC $\mathcal{M}_M = \langle S, \dashrightarrow, s, \mathscr{A} \rangle$, consider the cumulative probability function $\gamma_M : S \times 2^S \mapsto \mathbb{R}_+$ that accumulates the transition rates from a state $s' \in S$ to a set of states $\mathscr{S} \subseteq S$:

$$\gamma_M(s', \mathscr{S}) = \sum_{s'' \in \mathscr{S}} \left\{\!\!\left[\lambda \mid s' \xrightarrow{\lambda} s'' \right]\!\!\right\}$$

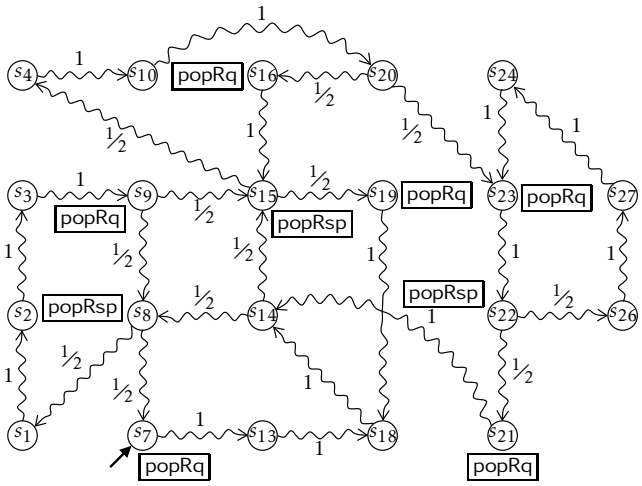


Figure 2.10: Minimized DTAMC of a 1-place FIFO queue DTMC model

Definition 2.35 (Markovian bisimulation)**.** *Given a CTAMC* $\mathcal{M}_M = \langle S, \dashrightarrow, s, \mathscr{A} \rangle$*, Markovian bisimulation equivalence* ($\sim_M$) *is the coarsest equivalence relation on* $\mathcal{M}_M$ *such that* $\left(\forall (s_1, s_2) \in S^2\right), s_1 \sim_M s_2$ *implies that for all equivalence class* $\mathcal{C}$ *of* $\sim_M$*,*

$$\begin{aligned} \gamma_M(s_1, \mathcal{C}) &= \gamma_M(s_2, \mathcal{C}) \\ \textit{and} \quad \mathscr{A}(s_1) &= \mathscr{A}(s_2) \end{aligned}$$

Two CTAMCs are Markovian bisimilar if their initial states are equivalent according to the CTAMC Markovian bisimulation ($\sim_p$).

Contrary to the discrete-time case, all states of a pure Markovian chain are not necessarily equivalent. However the annotations avoid aggregating equivalent states in the Markovian chain that do not have the same annotation.

As for DTAMCs, the Markovian bisimulation $\sim_M$ can be used to get the smallest CTAMC (in terms of state space size) equivalent to an other. This smallest equivalent CTAMC is called the *quotient* and we say that a CTAMC is minimal if it is of the size of its quotient.

Definition 2.36 (CTAMC quotient)**.** *Given a CTAMC* $\mathcal{M}_M = \langle S, \dashrightarrow, s, \mathscr{A} \rangle$*, the quotient of* $\mathcal{M}_M$ *is the DTAMC* $\mathcal{M}_{M/_{\sim_M}}$ *defined over the state space* $S_{/_{\sim_M}}$ *satisfying:*

- $\mathcal{M}_{M/_{\sim_M}}$ *and* $\mathcal{M}_M$ *are equivalent* ($\mathcal{M}_{M/_{\sim_M}} \sim_M \mathcal{M}_M$)
- *for every DTAMC* $\mathcal{M}'_M$*, defined over the state space* S'*, and equivalent to* $\mathcal{M}_M$ *(and to* $\mathcal{M}_{M/_{\sim_M}}$*), we have* $\mathsf{card}(S_{/_{\sim_M}}) \leq \mathsf{card}(S')$

Example 2.17. *The CTAMC depicted in figure 2.9 is minimal. Moreover, if we do not consider the annotations of this CTAMC (we just consider the pure CTMC depicted in figure 2.6), we have still a minimal CTAMC (or CTMC). Now, consider that the rates are all equals, i.e.,* $\delta_1 = \delta_2 = \lambda = \mu$*. With those settings, the CTAMC is no longer minimal. Its quotient is depicted in figure 2.11.*

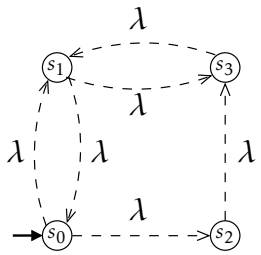


Figure 2.11: Minimized Markovian chain of a 1-place FIFO queue DTMC model

2.3 Semi-Markov Chains

A semi-Markov chain (SMC) [Kul95, Cin75] is an extension of a Markov chain. In an SMC, sojourn times in states are governed by general distributions, compared to exponentially distributed sojourn times in a CTMC and geometrically distributed sojourn times in a DTMC. In this section, we introduce the principal concepts of SMCs that are used in the following chapters. If transitions of SMC are labeled, they are also called labelled SMC, as in [LHK01]. Labeled SMC is the formalism we adopt throughout this thesis, but we simply talk about SMC.

Definition 2.37 (Semi-Markov chain)**.** *An SMC is a tuple* $\langle S, \mathcal{A}, \boxdot\!\!\rightsquigarrow, \mathcal{D}, s_0\rangle$ *where:*

- S *is a non empty set of states*
- $\mathcal{A}$ *is a set of actions*
- $\boxdot\!\!\rightsquigarrow : S\times\mathcal{A}\times]0,1]\times S$ *is a multiset of labeled probabilistic transitions such that for every state, probabilities of outgoing transitions sum up to 1, i.e.,* $\left(\forall s \in S\right) \sum_{s' in S \wedge a\in\mathcal{A}} \left\{\!\!\left\{ p \mid s \overset{a}{\boxdot\!\!\rightsquigarrow}{}^{p} s' \right\}\!\!\right\} = 1$
- $\mathcal{D} : S \times \mathcal{A} \times]0,1] \times S \mapsto (\mathbb{R}_+ \mapsto]0,1])$ *is a function that associates to each labeled probabilistic transition a general probability distribution function*
- s_0 *is the initial state*

Intuitively, in a state s of an SMC, the choice of a successor of s is performed probabilistically according to the probabilistic distribution given by the set of probabilistic transitions that can be taken by s. The sojourn time in s then follows the distribution associated to the chosen transition. For every SMC, it is possible to associate a DTMC, abstracting from actions and probability distribution functions associated to transitions. This DTMC is generally known as the *embedded DTMC* of the SMC. A formal definition of SMCs and embedded DTMCs can be found in [LHK01].

Definition 2.38 (Embedded DTMC of an SMC)**.** *Consider an SMC* $\mathcal{M}_S = \langle S, \mathcal{A}, \boxdot\!\!\rightsquigarrow, \mathcal{D}, s_0\rangle$*. The embedded DTMC of* $\mathcal{M}_S$ *characterized by the probabilistic chain* $\mathcal{M}_P = \langle S, \rightsquigarrow, s_0\rangle$ *satisfies:*

$$\left(\forall (s,s') \in S^2\right)\left(\forall a \in \mathcal{A}\right)\left(\forall p \in]0,1]\right) \quad \left(s \overset{a}{\boxdot\!\!\rightsquigarrow}{}^{p} s'\right) \Longrightarrow \left(s \overset{p}{\rightsquigarrow} s'\right)$$

In an SMC, as for Markov chains, we can study the average fraction of time $\widetilde{\pi}_s$ spent in each state s on the long run. According to [LHK01], the long-run average vector $\widetilde{\pi} = [\widetilde{\pi}_s]_{s\in S}$ can be computed from the steady-state probability vector of the embedded DTMC of the SMC (recall that when talking about the steady-state probability vector of a DTMC, we refer to its long-run average vector of transient probabilities, cf. 2.1.4). To provide a definition to $\widetilde{\pi}$, we first need to introduce some notions.

Consider an SMC $\mathcal{M}_S = \langle S, \mathcal{A}, \ \square\!\!\rightsquigarrow, \mathcal{D}, s_0 \rangle$. For every state $s \in S$, the probability distribution of the sojourn time in s, $\mathcal{D}_{\mathsf{SJ}} : S \times \mathbb{R}_+ \mapsto [0,1]$, (i.e., the distribution of the time spent in s before moving, each time s is reached) is defined by

$$\Big(\forall s \in S\Big)\ \Big(\forall t \in \mathbb{R}_+\Big) \quad \mathcal{D}_{\mathsf{SJ}}(s,t) = \sum_{s' \in S} \left\{\!\!\left[p \times \mathcal{D}(s,a,p,s')(t) \mid s \overset{a}{\square\!\!\rightsquigarrow}{}^{p} s' \right]\!\!\right\}$$

For convenience, we write $\mathcal{D}(s \overset{a}{\square\!\!\rightsquigarrow}{}^{p} s')$ instead of $\mathcal{D}(s,a,p,s')$. To define $\widetilde{\pi}$, we assume as [LHK01] that the system will stay in each state with at least some non-zero probability, and that the expected value $E\Big[\mathcal{D}_{\mathsf{SJ}}(s,.)\Big]$ of each sojourn time distribution is finite.

Definition 2.39 (long-run average of time spent in states of an SMC)**.** *Consider the SMC $\mathcal{M}_S$, $\mathcal{M}_S = \langle S, \mathcal{A}, \ \square\!\!\rightsquigarrow, \mathcal{D}, s_0 \rangle$, its embedded DTMC $\mathcal{M}_P$, $\mathcal{M}_P = \langle S, \ \rightsquigarrow, s_0 \rangle$, and the steady state probability vector $\pi = [\pi_i]_{i \in S}$ of $\mathcal{M}_P$. For every state $s \in S$, the long-run average $\widetilde{\pi}_s$ of time spent in s is given by:*

$$\widetilde{\pi}_s = \frac{\pi_s \, E\Big[\mathcal{D}_{\mathsf{SJ}}(s,.)\Big]}{\sum_{s' \in S} \Big(\pi_{s'} \, E\Big[\mathcal{D}_{\mathsf{SJ}}(s',.)\Big]\Big)}$$

2.4 Phase-Type Distributions

In this section we introduce a subclass of Markov chains allowing to depict a wide range of probability distributions.

Definition 2.40 (Discrete phase-type distribution)**.** *Consider a DTMC such that all states are transient but one that is absorbing. The distribution of the time needed, from time* 0*, to reach an absorbing state is a discrete phase-type distribution.*

Definition 2.41 (Continuous phase-type distribution)**.** *Consider a CTMC such that all states are transient but one that is absorbing. The distribution of the time needed, from time* 0*, to reach an absorbing state is a continuous phase-type distribution.*

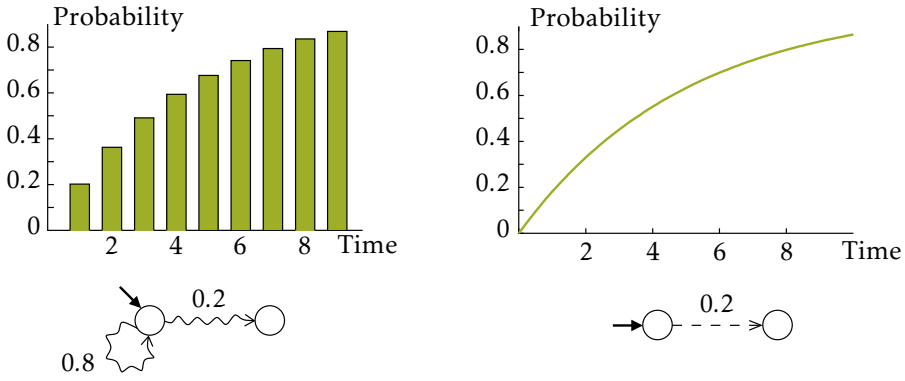
Phase-type distributions present a very interesting property [Neu81]:
- The class of discrete phase-type distributions is dense in the field of all discrete-time probability distributions
- The class of continuous phase-type distributions is dense in the field of all continuous-time probability distributions

This property implies that any discrete-time (resp. continuous-time) probability distribution can be fit arbitrary close by a phase-type distribution.

Example 2.18. *The simplest discrete and continuous phase-type distributions are the geometric distribution and the exponential distribution. Those distributions are depicted in figure 2.12 with there associated absorbing Markov chain.*

2.5 Discussion

Markov chains are a simple formalism to depict systems with time information. Nevertheless, for a large system, the construction of the Markov chain model remains a tedious task,



geometric distribution exponential distribution

Figure 2.12: Exponential and geometric distributions are phase-type distributions

reserved to a small group of experts. In addition, a direct modeling of complex systems with Markov chains is generally based on assumptions that have no formal justification. The accuracy of models depends on the expertise of people in charge of the modeling. To construct large Markov chains, a compositional approach relying on the assembly of smaller subcomponents models of a manageable complexity, would increase our confidence in the accuracy of the models.

When considering timed systems, one has to model time aspects like delays of the systems. Phase-type distributions are an interesting application of Markov chains. Theoretically, they can be used to approximate any kind of time distributions. Consequently, we should be able to model any kind of timed model using Markov chains (in discrete settings or in continuous settings).

Markov chains have been widely studied, and established performance analysis methods exist. Those performance analysis methods rely on functional information associated to the states of Markov chains. Functional information is in most cases directly taken into account in the definition of Markov chains. For the sake of clarity, we preferred to distinguish Markov chains from Markov chains with functional information, defining the annotated Markov chains. We illustrated the use of AMCs by associating to each state a reward, constructed on annotations. The notion of reward is one of the established approach to analyze Markov chains. In the next section, we define a complex performance measure on DTMCs, the latency, that can be seen as an application of the reward notion.

Chapter 3

Distribution of Latency in a DTMC

In the previous chapter, we saw that complex performance measures, based on state probabilities of a Markov chain, are a suitable target for studying complex systems. In this chapter, we formalize the notion of latency for DTMCs. It is a generalization of the latency defined in [CHLS09]. We define latency as a random variable denoting the number of time steps required, from one set of states, to observe a given number of times states of another set. We then provide solutions computing the long-run average of latency distributions. The latency distribution can be classified in the measures computed as a weighted sum of steady-state probabilities. We illustrate the study of two latencies in a FIFO queue.

3.1 Preliminaries

We introduce some notions concerning measure theory, inspired from [KNP07, Seg95]. The following definitions, lemmas and their proofs can be found in books dealing with measure theory as [Bog07, Bil95, Coh80, KSK76].

Definition 3.1 (σ-algebra)**.** *Consider an arbitrary non-empty set Ω and a family $\mathcal{F}$ of subsets of Ω, $\mathcal{F} \subseteq 2^{\Omega}$. $\mathcal{F}$ is a field on Ω if:*

- *the set Ω is in $\mathcal{F}$*
- *$\mathcal{F}$ is closed under complementation, $\left(\forall \mathcal{S} \in \mathcal{F}\right) \quad \Omega \backslash \mathcal{S} \in \mathcal{F}$*
- *$\mathcal{F}$ is closed under finite union, $\left(\forall \mathcal{S}_0, \ldots, \mathcal{S}_n \in \mathcal{F}\right) \quad \cup_{i=0}^{n} \mathcal{S}_i \in \mathcal{F}$*

A field $\mathcal{F}$ is a σ-algebra if it is also closed under countable union, $\left(\forall \{\mathcal{S}_i\}_{i \in \mathbb{N}} \quad \mathcal{S}_i \in \mathcal{F}\right) \quad \cup_{i \in \mathbb{N}} \mathcal{S}_i \in \mathcal{F}$.

The elements of a σ-algebra are called *measurable sets*, and the pair $(\Omega, \mathcal{F})$ is called a *measurable space*. Considering an arbitrary non-empty set Ω and a family $\mathcal{F}$ of subsets of ω, $\mathcal{F} \subseteq 2^{\Omega}$, there exists a unique smallest σ-algebra containing $\mathcal{F}$. It is called the σ-algebra *generated* by the family $\mathcal{F}$ and we write it $\Sigma_{\mathcal{F}}$.

Definition 3.2 (measure)**.** *Consider a measurable space $(\Omega, \mathcal{F})$. A function $\mu : \mathcal{F} \mapsto [0, +\infty[$ is a measure on $(\Omega, \mathcal{F})$ if:*

- *μ is null for the empty set, $\mu(\varnothing) = 0$*
- *μ is positive for every set, $\left(\forall \mathcal{S} \in \mathcal{F}\right) \quad \mu(\mathcal{S}) \geq 0$*
- *For all countable families $\{\mathcal{S}_i\}_{i \in I \subseteq \mathbb{N}}$ of pairwise disjoint sets in $\mathcal{F}$, we have $\mu(\cup_{i \in I} \mathcal{S}_i) = \sum_{i \in I} \mu(\mathcal{S}_i)$*

If μ is a measure on the measurable space $(\Omega, \mathcal{F})$, the tuple $(\Omega, \mathcal{F}, \mu)$ is referred to as a *measure space*.

Definition 3.3 (probability measure)**.** *Consider a measurable space* $(\Omega, \mathcal{F})$*. A function* $\mu : \mathcal{F} \mapsto [0,1]$ *is a probability measure on* $(\Omega, \mathcal{F})$ *if:*

- μ *is a measure on* $(\Omega, \mathcal{F})$
- μ *is equal to one for* Ω*,* $\mu(\Omega) = 1$

If μ is a probability measure on the measurable space $(\Omega, \mathcal{F})$, the tuple $(\Omega, \mathcal{F}, \mu)$ is referred to as a *probability space*, where the set Ω is called the *sample space*, and the elements of $\mathcal{F}$ are called *events*.

Definition 3.4 (semi-ring)**.** *Consider an arbitrary non-empty set* Ω *and a family* $\mathcal{F}$ *of subsets of* Ω*,* $\mathcal{F} \subseteq 2^{\Omega}$*.* $\mathcal{F}$ *is a semi-ring if:*

(1) the empty set $\varnothing$ *is in* $\mathcal{F}$

(2) for all sets $\mathcal{S}_1, \mathcal{S}_2 \in \mathcal{F}$*, their intersection is in* $\mathcal{F}$*,* $\mathcal{S}_1 \cap \mathcal{S}_2 \in \mathcal{F}$

(3) for all sets $\mathcal{S}_1, \mathcal{S}_2 \in \mathcal{F}$ *with* $\mathcal{S}_2 \subseteq \mathcal{S}_1$*, the set* $\mathcal{S}_1 \backslash \mathcal{S}_2$ *is the union of finitely many disjoint sets* $\mathcal{S}_1', \ldots, \mathcal{S}_k'$ *in* $\mathcal{F}$*,* $\mathcal{S}_1 \backslash \mathcal{S}_2 = \cup_{i=1}^{k} \mathcal{S}_i'$

Lemma 3.1 (Measure on the σ-algebra generated by a semi-ring)**.** *Consider a semi-ring* $\mathcal{F}$ *on an arbitrary non-empty set* Ω*, and a function* $\mu : \mathcal{F} \mapsto [0, +\infty[$ *satisfying:*

- $\mu(\varnothing) = 0$
- *For all families* $\{\mathcal{S}_1, \ldots, \mathcal{S}_k\}$ *of pairwise disjoint sets in* $\mathcal{F}$*,* $\mu\left(\cup_{i=1}^{k} \mathcal{S}_i\right) = \sum_{i=1}^{k} \mu(\mathcal{S}_i)$
- *For all countable families* $\{\mathcal{S}_i\}_{i \in I \subseteq \mathbb{N}}$ *of pairwise disjoint sets in* $\mathcal{F}$*,* $\mu(\cup_{i \in I} \mathcal{S}_i) \leq \sum_{i \in I} \mu(\mathcal{S}_i)$

The function μ *extends to a unique measure on the* σ*-algebra generated by* $\mathcal{F}$*.*

3.2 Latency Definition

Consider a homogeneous DTMC $X = \{X_n, n \in \mathbb{N}\}$ defined over a state space S and initially in a state $\overline{s}$. X is characterized by the probabilistic chain $\mathcal{M}_P = \langle S, \rightsquigarrow, \overline{s} \rangle$. We use X^s to denote the DTMC characterized by $\mathcal{M}_P^s = \langle S, \rightsquigarrow, s \rangle$, i.e., the DTMC where the initial state is s instead of $\overline{s}$.

Recall that in the DTMC X, the transition probability $p_{s_i s_j}$ from a state $s_i \in S$ to a state $s_j \in S$ is obtained by summing over all the possible transitions $s_i \rightsquigarrow s_j$ in $\mathcal{M}_P$:

$$\left(\forall (s_i, s_j) \in S^2\right) \quad p_{s_i s_j} = \sum \left\{ \left[p \mid s_i \overset{p}{\rightsquigarrow} s_j \right] \right\}$$

An execution of the DTMC X is represented by a *path* in $\mathcal{M}_P$. A path $\sigma = s_0\ s_1\ s_2\ \cdots$ is a non-empty sequence of states in S, satisfying:

$$\left(\forall i \geq 0\right) \quad s_i \rightsquigarrow s_{i+1}$$

A path $\sigma = s_0\ s_1\ s_2\ \cdots$ can either be finite or infinite and we let $\sigma@i$ denote the ith state of the path σ, i.e., $\sigma@i = s_i$. For a finite path, we write $\vec{\sigma}$. $last(\vec{\sigma})$ is the last state of $\vec{\sigma}$ and $|\vec{\sigma}|$ is the length of $\vec{\sigma}$. For instance, for a finite path $\vec{\sigma} = s_0\ s_1\ s_2\ s_3\ s_4$ we have $last(\vec{\sigma}) = s_4$ and $|\vec{\sigma}| = 4$. The

length of a finite path is not the number of states along the path, but the number of transitions. A finite path $\vec{\sigma}$ of length n is a *prefix* of an infinite path σ if

$$\Big(\forall i\Big)\,(0 \leq i \leq n) \quad \vec{\sigma}@i = \sigma@i$$

$Paths(s)$ and $Paths^{\vec{}}(s)$ denote respectively the set of all infinite and finite paths starting in state s of S.

Firstly, we define a probability associated with a finite path, by multiplying successive transition probabilities along the path.

Definition 3.5 (Probability of a finite path)**.** *For every finite path $\vec{\sigma} \in Paths^{\vec{}}(s)$ of length n ($|\vec{\sigma}| = n$) we define the probability $P^s\{\vec{\sigma}\}$:*

$$P^s\{\vec{\sigma}\} = \begin{cases} 1 & \text{if } n = 0 \\ \prod\limits_{1 \leq i < n} p_{\sigma[i]\sigma[i+1]} & \text{otherwise} \end{cases}$$

Secondly, we define the cylinder set $C(\vec{\sigma}) \subseteq Paths(s)$ as the set of all infinite paths with prefix $\vec{\sigma}$.

Definition 3.6 (Cylinder set induced by a finite path)**.** *The cylinder set $C(\vec{\sigma}) \subseteq Paths(s)$ is defined as:*

$$C(\vec{\sigma}) = \{\sigma \in Paths(s) \mid \vec{\sigma} \text{ is a prefix of } \sigma\}$$

We first prove that the set $\mathcal{F}$ of cylinder sets $C(\vec{\sigma})$ where $\vec{\sigma}$ ranges over the finite paths $Paths^{\vec{}}(s)$ with the empty set $\emptyset$ form a semi-ring.

Lemma 3.2 (Cylinder sets form a semi-ring)**.** *The set $\mathcal{F}$ of cylinder sets and the empty set, $\mathcal{F} = \emptyset \cup \{C(\vec{\sigma})\}_{\vec{\sigma} \in Paths^{\vec{}}(s)}$ form a semi-ring over $Paths(s)$.*

Proof. Firstly, we prove that the intersection of two sets of $\mathcal{F}$ is in $\mathcal{F}$ (property *(2)* of definition 3.4). Consider two cylinder sets C_1 and C_2 in $\mathcal{F}$. C_1 and C_2 are generated by two finite paths $\vec{\sigma}_1$ and $\vec{\sigma}_2$. We can distinguish several possibilities:

– Consider that the two finite paths differ :

$$\Big(\exists i \in \mathbb{N}\; i \leq |\vec{\sigma}_1| \wedge i \leq |\vec{\sigma}_2|\Big) \quad \vec{\sigma}_1@i \neq \vec{\sigma}_2@i$$

In this case, the intersection of the two cylinder sets is empty $C_1 \cap C_2 = \emptyset$ and is thus in $\mathcal{F}$, $\emptyset \in \mathcal{F}$.

– Or Consider that one of the finite paths is a prefix of the second one. We tackle the case for which $|\vec{\sigma}_1| \leq |\vec{\sigma}_2|$:

$$\Big(\forall i \in \mathbb{N}\; i \leq |\vec{\sigma}_1|\Big) \quad \vec{\sigma}_1@i = \vec{\sigma}_2@i$$

We can deduce that $C_2 \subseteq C_1$ and consequently $C_1 \cap C_2 = C_2$, which is also in $\mathcal{F}$, $C_1 \cap C_2 \in \mathcal{F}$.

Secondly, we prove that for two cylinder sets C_1 and C_2 with $C_2 \subseteq C_1$, $C_1 \backslash C_2$ is the union of finitely many disjoint sets of $\mathcal{F}$ (property *(3)* of definition 3.4). if $C_1 = C_2$, we have $C_1 \backslash C_2 = \emptyset$ and the result is obvious. Now, consider that $C_2 \subset C_1$. We have $|\vec{\sigma}_1| \leq |\vec{\sigma}_2|$ and

$$\Big(\forall i \in \mathbb{N}\; i < |\vec{\sigma}_1|\Big) \quad \vec{\sigma}_1@i = \vec{\sigma}_2@i.$$

We can thus write that :

$$
\begin{aligned}
C_1 \backslash C_2 &= \{\sigma \in Paths(s) \mid \vec{\sigma}_1 \text{ is a prefix of } \sigma \text{ and } \vec{\sigma}_2 \text{ is not a prefix of } \sigma\} \\
&= \left\{\sigma \in Paths(s) \mid \vec{\sigma}_1 \text{ is a prefix of } \sigma \wedge \left(\exists i \in \mathbb{N} \; |\vec{\sigma}_1| \leq i < |\vec{\sigma}_2|\right) \quad \sigma@i \neq \vec{\sigma}_2@i\right\} \\
&= \bigcup_{i=|\vec{\sigma}_1|+1}^{|\vec{\sigma}_2|} \{\sigma \in Paths(s) \mid \vec{\sigma}_1 \text{ is a prefix of } \sigma \wedge \sigma@i \neq \vec{\sigma}_2@i\} \\
&= \bigcup_{i=|\vec{\sigma}_1|+1}^{|\vec{\sigma}_2|} \bigcup_{\left\{\vec{\sigma}' \;\middle|\; \begin{array}{l} |\vec{\sigma}'| = |\vec{\sigma}_2| \quad \wedge \\ \left(\forall k \leq |\vec{\sigma}_1|\right) \; \vec{\sigma}'@k = \vec{\sigma}_1@k \\ \wedge \quad \vec{\sigma}'@i \neq \vec{\sigma}_2@i \end{array}\right\}} \{\sigma \in Paths(s) \mid \vec{\sigma}' \text{ is a prefix of } \sigma\}
\end{aligned}
$$

The set $\{\sigma \in Paths(s) \mid \vec{\sigma}' \text{ is a prefix of } \sigma\}$ is a cylinder set and the union is finite because of the finite state space of the DTMC.

□

Then we define a function $\Pr^s : \mathcal{F} \mapsto [0,1[$ over the semi-ring $\mathcal{F}$:

$$
\left(\forall \vec{\sigma} \in Path\vec{s}(s)\right) \quad \begin{cases} \Pr^s\{C(\vec{\sigma})\} = P^s(\vec{\sigma}) \\ \Pr^s\{\emptyset\} = 0 \end{cases}
$$

satisfying the following properties:

– For all families $\{C_1, \ldots, C_k\}$ of pairwise disjoint sets in $\mathcal{F}$, $\Pr^s\left(\cup_{i=1}^k C_i\right) = \sum_{i=1}^k \Pr^s(C_i)$.

– For all countable families $\{C_i\}_{i \in I \subseteq \mathbb{N}}$ of pairwise disjoint sets in $\mathcal{F}$, $\Pr^s(\cup_{i \in I} C_i) \leq \sum_{i \in I} \Pr^s(C_i)$.

According to lemma 3.1, the function $\Pr^s$ is a probability measure on the σ-algebra generated by $\mathcal{F}$. We call this σ-algebra $\Sigma_{Paths(s)}$.

Let us consider the σ-algebra $\Sigma_{Paths(\overline{s})}$ defined from the initial state $\overline{s}$, and the corresponding probability measure $\Pr^{\overline{s}}$. On the basis of $\Pr^{\overline{s}}$, we can define various probability measures such as the transient state probabilities in the DTMC.

Definition 3.7 (Transient state probabilities)**.** *The probability of being in state s at time n_0 is given by*

$$
\pi_s(n_0) = \Pr^{\overline{s}}\{\sigma \in Paths(\overline{s}) | \sigma@n_0 = s\}
$$

To make the notion of latency precise, we identify two sets of states $\alpha \subseteq S$ and $\omega \subseteq S$ and a function $\mathsf{k} : \alpha \mapsto \mathbb{N}_+$, over the set of states α. The latency corresponds to the number of time-steps required, from a state $s \in \alpha$, to observe $\mathsf{k}(s)$-times a state of ω, $\mathsf{k}(s)$ depending on the state $s \in \alpha$. $\mathsf{k}(s)$ denotes the number of observations of a state of ω defining the latency starting in s. The latency $\mathcal{L}_{(\alpha,\omega,\mathsf{k})}$ is a function over time and the paths starting in the initial state, $\mathcal{L}_{(\alpha,\omega,\mathsf{k})} : Paths(\overline{s}) \times \mathbb{N} \mapsto \mathbb{N} \cup \perp$, which associates the value of the latency in a path σ starting at a given time n_0. $\mathcal{L}_{(\alpha,\omega,\mathsf{k})}(\sigma, n_0)$ is undefined ($\perp$) if the state $\sigma@n_0 \notin \alpha$, i.e., if $\sigma@n_0$ does not correspond to a starting state for the latency.

Definition 3.8 (Latency at time n_0 in a path)**.** *For every path σ starting in the initial state of the*

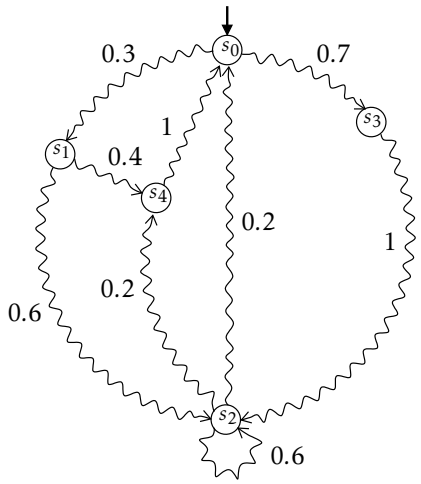


Figure 3.1: Markov chain $\mathcal{M}_P$ illustrating the latency measure

Markov chain, $\sigma \in Paths(\overline{s})$, we have:

$$\mathcal{L}_{(\alpha,\omega,\mathsf{k})}(\sigma, n_0) = \begin{cases} \bot & \text{if} \quad \sigma@n_0 \notin \alpha \\ n & \text{if} \quad s = \sigma@n_0 \in \alpha \\ & \text{and} \quad \left(\exists \mathscr{S} = \left\{n'_1, n'_2, n'_{\mathsf{k}(s)}\right\}\right) \quad \left(\begin{array}{l} 0 \leq n'_1 < \cdots < n'_{\mathsf{k}(s)} = n \quad \wedge \quad \left(\forall n' \leq n\right) \\ (n' \in \mathscr{S}) \Longleftrightarrow (\sigma@(n_0 + n') \in \omega) \end{array} \right) \\ \infty & \text{otherwise} \end{cases}$$

The latency starting at time n_0 of a path σ is potentially infinite if $\sigma@n_0 \in \alpha$ and it is not possible to observe $\mathsf{k}(s)$-times a state of ω in the the path after time n_0.

Example 3.1. *Consider the DTMC $\mathcal{M}_P$ of Fig. 3.1 and sets $\alpha = \{s_1, s_3\}$ and $\omega = \{s_0\}$. The values $\mathsf{k}(s_1)$ and $\mathsf{k}(s_3)$ are defined such that $\mathsf{k}(s_1) = \mathsf{k}(s_3) = 1$, i.e., the considered latency corresponds to the observation of the first occurrence of a state of ω, for every starting state of α.*

We list some paths illustrating the latency value:

- *Every path σ with prefix $(s_0\, s_1\, s_4\, s_0)$ satisfies $\mathcal{L}_{(\alpha,\omega,\mathsf{k})}(\sigma, 1) = 2$ ($\sigma@1 = s_1 \in \alpha$, $\sigma@(1+2) = s_0 \in \omega$ and the state of the path between time 1 and time 3 (i.e., at time 2) is $s_4 \notin \omega$).*
- *Every path σ with prefix $(s_0\, s_1\, s_4\, s_0\, s_3\, s_2\, s_2\, s_2\, s_0)$ satisfies $\mathcal{L}_{(\alpha,\omega,\mathsf{k})}(\sigma, 4) = 4$ ($\sigma@4 = s_3 \in \alpha$, $\sigma@(4+4) = s_0 \in \omega$, and the states of the path between time 4 and time 8 are not in ω).*
- *Every path σ with prefix $(s_0\, s_3\, s_2\, s_2\, s_4)$ satisfies $\mathcal{L}_{(\alpha,\omega,\mathsf{k})}(\sigma, 2) = \bot$ ($\sigma@2 = s_2 \notin \alpha$).*

On the basis of the probability measure $\mathsf{Pr}^{\overline{s}}$, we define the probability distribution of the latency at time n_0, defined as following.

Definition 3.9 (Probability distribution of the latency at time n_0)**.**

$$\left(\forall n \in \mathbb{N} \cup \bot\right) \quad \mathsf{Pr}^{n}_{\mathcal{L}_{(\alpha,\omega,\mathsf{k})}}(n_0) = \mathsf{Pr}^{\overline{s}}\left\{\sigma \in Paths(\overline{s}) \,\middle|\, \mathcal{L}_{(\alpha,\omega,\mathsf{k})}(\sigma, n_0) = n\right\}$$

One may be interested in the value of $\mathsf{Pr}^{n}_{\mathcal{L}_{(\alpha,\omega,\mathsf{k})}}(n_0)$ when time n_0 tends to infinity, i.e., the steady state distribution of the latency. Unfortunately, the steady state distribution of the latency may not exist. However, we can define a related measure representing a long-run average of latency distributions.

Let σ be a path randomly taken in $Paths(\overline{s})$. We define the random variable $\mathbf{1}_{[n]}\left(\mathcal{L}_{(\alpha,\omega,\mathsf{k})}(\sigma,n_0)\right)$ that indicates whether $\mathcal{L}_{(\alpha,\omega,\mathsf{k})}(\sigma,n_0)=n$ (in this case, it is equal to one) or not (in this case, it is equal to zero):

$$\left(\forall n_0\in\mathbb{N}\right)\left(\forall n\in\mathbb{N}\cup\bot\right)\left(\forall\sigma\in Paths(\overline{s})\right)\quad \mathbf{1}_{[n]}\left(\mathcal{L}_{(\alpha,\omega,\mathsf{k})}(\sigma,n_0)\right)=\begin{cases}1 & \text{if } \mathcal{L}_{(\alpha,\omega,\mathsf{k})}(\sigma,n_0)=n\\ 0 & \text{otherwise}\end{cases}$$

We can then define the random variable Φ_{n,n_0}, which totals the number of times the latency was equal to n in σ before time n_0 and normalized by n_0. This random variable is defined as follows:

$$\Phi_{n,n_0}=\frac{1}{n_0}\sum_{i=0}^{n_0}\mathbf{1}_{[n]}\left(\mathcal{L}_{(\alpha,\omega,\mathsf{k})}(\sigma,n_0)\right)$$

The expected value $E\left[\Phi_{n,n_0}\right]$ of the random variable Φ_{n,n_0} corresponds to the average fraction of time the latency is equal to n before time n_0. The limit, when n_0 tends to infinity, of this expected value, defines the long-run average probability distribution of the latency

Definition 3.10 (Long-run average probability distribution of the latency)**.** *The long-run average fraction of time* $\overline{\mathsf{Pr}}^n_{\mathcal{L}_{(\alpha,\omega,\mathsf{k})}}$ *the latency is equal to n is given by*

$$\left(\forall n\in\mathbb{N}\cup\bot\right)\quad \overline{\mathsf{Pr}}^n_{\mathcal{L}_{(\alpha,\omega,\mathsf{k})}}=\lim_{n_0\to\infty}E\left[\Phi_{n,n_0}\right]=\lim_{n_0\to\infty}E\left[\frac{1}{n_0}\sum_{i=0}^{n_0}\mathbf{1}_{[n]}\left(\mathcal{L}_{(\alpha,\omega,\mathsf{k})}(\sigma,n_0)\right)\right]$$

where σ *ranges randomly over* $Paths(\overline{s})$.

The long-run average distribution of the latency is consequently given by the distribution $\overline{\mathsf{Pr}}^n_{\mathcal{L}_{(\alpha,\omega,\mathsf{k})}}$ with $n\in\mathbb{N}\cup\bot$.

Lemma 3.3.

$$\left(\forall n\in\mathbb{N}\cup\bot\right)\quad \overline{\mathsf{Pr}}^n_{\mathcal{L}_{(\alpha,\omega,\mathsf{k})}}=\lim_{n_0\to\infty}\frac{1}{n_0}\sum_{i=0}^{n_0}\mathsf{Pr}^{\overline{s}}\left\{\sigma\in Paths(\overline{s})\mid \mathcal{L}_{(\alpha,\omega,\mathsf{k})}(\sigma,n_0)=n\right\}$$

Proof. By direct computation of the expected value, we have for every value $n\in\mathbb{N}\cup\bot$:

$$\begin{aligned}
\overline{\mathsf{Pr}}^n_{\mathcal{L}_{(\alpha,\omega,\mathsf{k})}} &= \lim_{n_0\to\infty}E\left[\frac{1}{n_0}\sum_{i=0}^{n_0}\mathbf{1}_{[n]}\left(\mathcal{L}_{(\alpha,\omega,\mathsf{k})}(\sigma,n_0)\right)\right]\\
&= \lim_{n_0\to\infty}\frac{1}{n_0}\sum_{i=0}^{n_0}E\left[\mathbf{1}_{[n]}\left(\mathcal{L}_{(\alpha,\omega,\mathsf{k})}(\sigma,n_0)\right)\right]\\
&= \lim_{n_0\to\infty}\frac{1}{n_0}\sum_{i=0}^{n_0}\left(1\times\mathsf{Pr}\left\{\mathbf{1}_{[n]}\left(\mathcal{L}_{(\alpha,\omega,\mathsf{k})}(\sigma,n_0)\right)=n\right\}+0\times\mathsf{Pr}\left\{\mathbf{1}_{[n]}\left(\mathcal{L}_{(\alpha,\omega,\mathsf{k})}(\sigma,n_0)\right)\neq n\right\}\right)\\
&= \lim_{n_0\to\infty}\frac{1}{n_0}\sum_{i=0}^{n_0}\mathsf{Pr}\left\{\mathbf{1}_{[n]}\left(\mathcal{L}_{(\alpha,\omega,\mathsf{k})}(\sigma,n_0)\right)=n\right\}\\
&= \lim_{n_0\to\infty}\frac{1}{n_0}\sum_{i=0}^{n_0}\mathsf{Pr}^n_{\mathcal{L}_{(\alpha,\omega,\mathsf{k})}}(n_0)\\
&= \lim_{n_0\to\infty}\frac{1}{n_0}\sum_{i=0}^{n_0}\mathsf{Pr}^{\overline{s}}\left\{\sigma\in Paths(\overline{s})\mid \mathcal{L}_{(\alpha,\omega,\mathsf{k})}(\sigma,n_0)=n\right\}
\end{aligned}$$

□

3.3 Computation of the Latency Distribution

In this section, we provide an easily-computable expression of the long-run average distribution of the latency.

Lemma 3.4. *The probability distribution of the latency at time n_0 satisfies:*

$$\left(\forall n \in \mathbb{N} \cup \bot\right) \quad \mathsf{Pr}^n_{\mathcal{L}_{(\alpha,\omega,\mathsf{k})}}(n_0) = \begin{cases} \displaystyle\sum_{s\in\alpha} \pi_s(n_0) \times \mathsf{Pr}^s\left\{\sigma \in \mathit{Paths}(s) \mid \mathcal{L}_{(\alpha,\omega,\mathsf{k})}(\sigma,0) = n\right\} & \text{if } n \in \mathbb{N} \\ \displaystyle 1 - \sum_{s\in\alpha} \pi_s(n_0) & \text{if } n = \bot \end{cases}$$

Proof. By direct calculus from definition 3.9. The case were the value n is equal to $\bot$ is direct: at time n_0, the latency is undefined if the occupied state is not in the set α.

For a value n that differs from $\bot$, we have:

$$\begin{aligned} \mathsf{Pr}^n_{\mathcal{L}_{(\alpha,\omega,\mathsf{k})}}(n_0) &= \mathsf{Pr}^{\overline{s}}\left\{\sigma \in \mathit{Paths}(\overline{s}) \mid \mathcal{L}_{(\alpha,\omega,\mathsf{k})}(\sigma,n_0) = n\right\} \\ &= \sum_{s\in S} \mathsf{Pr}^{\overline{s}}\left\{\sigma \in \mathit{Paths}(\overline{s}) \mid \sigma@n_0 = s \wedge \mathcal{L}_{(\alpha,\omega,\mathsf{k})}(\sigma,n_0) = n\right\} \\ &= \sum_{s\in S} \mathsf{Pr}^{\overline{s}}\{\sigma \in \mathit{Paths}(\overline{s}) \mid \sigma@n_0 = s\} \times \mathsf{Pr}^s\left\{\sigma \in \mathit{Paths}(s) \mid \mathcal{L}_{(\alpha,\omega,\mathsf{k})}(\sigma,0) = n\right\} \\ &= \sum_{s\in S} \pi_s(n_0) \times \underbrace{\mathsf{Pr}^s\left\{\sigma \in \mathit{Paths}(s) \mid \mathcal{L}_{(\alpha,\omega,\mathsf{k})}(\sigma,0) = n\right\}}_{0 \text{ if } (\sigma@0)\notin\alpha} \\ &= \sum_{s\in\alpha} \pi_s(n_0) \times \mathsf{Pr}^s\left\{\sigma \in \mathit{Paths}(s) \mid \mathcal{L}_{(\alpha,\omega,\mathsf{k})}(\sigma,0) = n\right\} \end{aligned}$$

□

Lemma 3.5. *The long-run average probability distribution of the latency satisfies:*

$$\left(\forall n \in \mathbb{N} \cup \bot\right) \quad \overline{\mathsf{Pr}}^n_{\mathcal{L}_{(\alpha,\omega,k)}} = \begin{cases} \displaystyle\sum_{s\in\alpha} \pi_s \times \mathsf{Pr}^s\left\{\sigma \in \mathit{Paths}(s) \mid \mathcal{L}_{(\alpha,\omega,k)}(\sigma,0) = n\right\} & \text{if } n \in \mathbb{N} \\ \displaystyle 1 - \sum_{s\in\alpha} \pi_s & \text{if } n = \bot \end{cases}$$

Proof. By direct calculus from definition 3.3. The case were the value n is equal to $\bot$ is obvious.

For a value n that differs from $\perp$, we have:

$$\begin{aligned}
\overline{\Pr}^{n}_{\mathcal{L}_{(\alpha,\omega,\mathsf{k})}} &= \lim_{n_0\to\infty}\frac{1}{n_0}\sum_{i=0}^{n_0}\Pr^{n}_{\mathcal{L}_{(\alpha,\omega,\mathsf{k})}}(n_0)\\
&= \lim_{n_0\to\infty}\frac{1}{n_0}\sum_{i=0}^{n_0}\left(\sum_{s\in\alpha}\pi_s(n_0)\times\Pr^{s}\left\{\sigma\in Paths(s)\mid \mathcal{L}_{(\alpha,\omega,\mathsf{k})}(\sigma,0)=n\right\}\right)\\
&= \sum_{s\in\alpha}\lim_{n_0\to\infty}\frac{1}{n_0}\sum_{i=0}^{n_0}\left(\pi_s(n_0)\times\underbrace{\Pr^{s}\left\{\sigma\in Paths(s)\mid \mathcal{L}_{(\alpha,\omega,\mathsf{k})}(\sigma,0)=n\right\}}_{\text{value independent from } n_0}\right)\\
&= \sum_{s\in\alpha}\underbrace{\left(\lim_{n_0\to\infty}\frac{1}{n_0}\sum_{i=0}^{n_0}\pi_s(n_0)\right)}_{\text{long-run average state probability}}\times\Pr^{s}\left\{\sigma\in Paths(s)\mid \mathcal{L}_{(\alpha,\omega,\mathsf{k})}(\sigma,0)=n\right\}\\
&= \sum_{s\in\alpha}\pi_s\times\Pr^{s}\left\{\sigma\in Paths(s)\mid \mathcal{L}_{(\alpha,\omega,\mathsf{k})}(\sigma,0)=n\right\}
\end{aligned}$$

□

Actually, we are interested in the long-run average probability distribution of the latency when it is defined, i.e., when the latency differs from $\perp$. Consequently, we normalize the distribution so as to keep only the values where the latency is defined.

Definition 3.11 (Normalized long-run average distribution of the latency)**.** *The normalized long-run average distribution,* $\widetilde{\Pr}$*, of the latency is defined by*

$$\left(\forall n\in\mathbb{N}\right)\quad \widetilde{\Pr}^{n}_{\mathcal{L}_{(\alpha,\omega,k)}}=\sum_{s\in\alpha}\frac{\pi_s}{\sum_{s'\in\alpha}\pi_{s'}}\times\Pr^{s}\left\{\sigma\in Paths(s)\mid \mathcal{L}_{(\alpha,\omega,\mathsf{k})}(\sigma,0)=n\right\}$$

We can see that the normalized long-run average distribution of the latency is a (normalized) weighted sum of the steady-state probabilities. For each state $s\in\alpha$, the weighting factor

$$\Pr^{s}\left\{\sigma\in Paths(s)\mid \mathcal{L}_{(\alpha,\omega,\mathsf{k})}(\sigma,0)=n\right\},$$

which is time-independent, characterizes the probability that the latency, from the state s, is equal to n. Additionally, we can say that, for a given state $s\in\alpha$, the set of all weighting factors where n ranges over $\mathbb{N}$ is also the distribution $\widetilde{\Pr}^{n}_{\mathcal{L}_{(\{s\},\omega,\mathsf{k})}}$ of the latency starting in s. This distribution can be computed considering the execution tree induced by the set of execution paths $\left\{\sigma\in Paths(s)\mid \mathcal{L}_{(\alpha,\omega,\mathsf{k})}(\sigma,0)=n\right\}_{n\in\mathbb{N}}$, for which the latency starting in s is defined.

Definition 3.12 (Execution tree induced by the set of execution paths for which the latency is defined)**.** *For each state $s\in\alpha$, the execution tree induced by the set of execution paths for which the latency is defined is an absorbing Markov chain $\mathcal{M}_P(s)$ starting in s. The absorbing Markov chain $\mathcal{M}_P(s)$ is constructed over the set of path $\mathcal{S}$ in $\mathcal{M}_P$, starting in s and such that the time dependent latency at time 0 is defined:*

$$\mathcal{S}=\left\{\sigma\in Paths(s)\mid \left(\exists n\in\mathbb{N}\right)\quad \mathcal{L}_{(\alpha,\omega,\mathsf{k})}(\sigma,0)=n\right\}$$

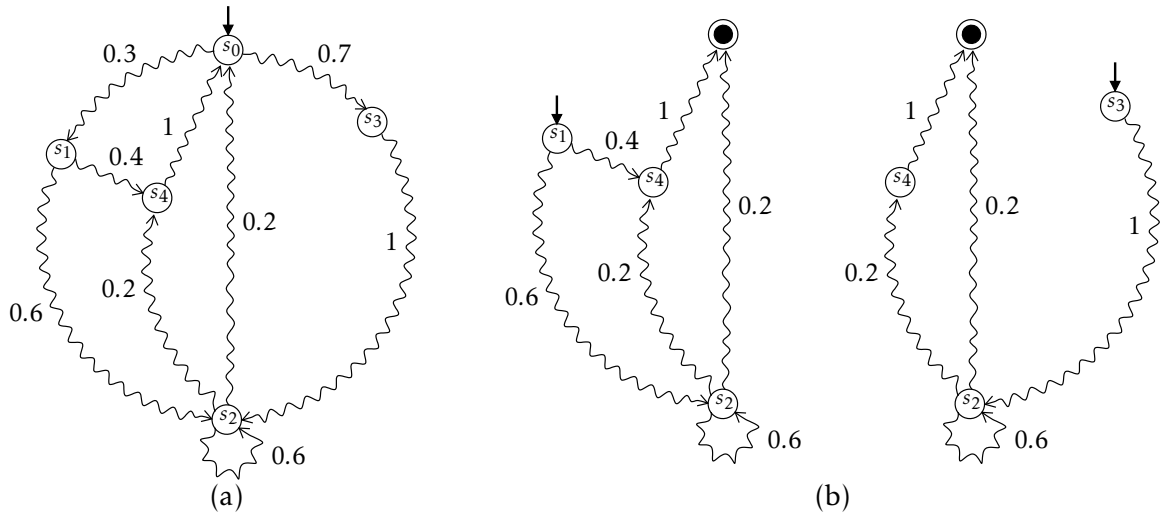


Figure 3.2: Markov chain $\mathcal{M}_P$ and extracted subchains $\mathcal{M}_P(s_1)$ and $\mathcal{M}_P(s_3)$

For each path $\sigma \in \mathcal{S}$, the prefix $\bar{\sigma}$ of length $|\bar{\sigma}| = n$ of σ is also a finite path in $\mathcal{M}_P(s)$, and $\bar{\sigma}@n$ is an absorbing state of $\mathcal{M}_P(s)$.

For each state $s \in \alpha$, the absorbing subchain $\mathcal{M}_P(s)$ can be used to compute the distribution $\widetilde{\mathsf{Pr}}^n_{\mathcal{L}_{(\{s\},\omega,\mathsf{k})}}$ by transient analysis: the distribution of the time needed to reach an absorbing state in $\mathcal{M}_P(s)$ is equal to the distribution $\widetilde{\mathsf{Pr}}^n_{\mathcal{L}_{(\{s\},\omega,\mathsf{k})}}$.

The computation of the normalized long-run average distribution of the latency depending only on steady-state and transient probabilities, the worst-case time complexity for computing latencies is determined by the time complexity of algorithms used to compute steady state and transient probabilities computations.

Example 3.2. *We consider the DTMC $\mathcal{M}_P$ and the studied latency of example 3.1 ($\mathcal{M}_P$ is drawn again in figure 3.2(a)). The parameters of the latency are: $\alpha = \{s_1, s_3\}$, $\omega = \{s_0\}$ and $\mathsf{k}(s_1) = \mathsf{k}(s_3) = 1$.*

We can construct two absorbing subchains, $\mathcal{M}_P(s_1)$ and $\mathcal{M}_P(s_3)$, starting in s_1 and s_3, which allow to compute the latency distributions starting in s_1 and in s_3, $\widetilde{\mathsf{Pr}}^n_{\mathcal{L}_{(\{s_1\},\omega,\mathsf{k})}}$ and $\widetilde{\mathsf{Pr}}^n_{\mathcal{L}_{(\{s_3\},\omega,\mathsf{k})}}$. $\mathcal{M}_P(s_1)$ and $\mathcal{M}_P(s_3)$ are depicted in figure 3.2(b), where the black-filled states are the absorbing states.

The latency distribution $\widetilde{\mathsf{Pr}}^n_{\mathcal{L}_{(\alpha,\omega,\mathsf{k})}}$ is depicted in figure 3.3. The distributions $\widetilde{\mathsf{Pr}}^n_{\mathcal{L}_{(\{s_1\},\omega,\mathsf{k})}}$ and $\widetilde{\mathsf{Pr}}^n_{\mathcal{L}_{(\{s_3\},\omega,\mathsf{k})}}$, weighted by their respective (normalized) steady state probabilities, $\frac{\pi_{s_1}}{\pi_{s_1}+\pi_{s_3}}$ and $\frac{\pi_{s_3}}{\pi_{s_1}+\pi_{s_3}}$, are also depicted in figure 3.3.

3.4 Latency Distribution in Practice

We presented a way to compute the distribution of a latency in the previous sections. The definition of a latency relies on the identification of two sets of states α and ω of the DTMC, and a function k that associates to each state of α a positive integer value ($\mathsf{k} : \alpha \mapsto \mathbb{N}_+$). In the previous section, we did not provide information on how those sets α and ω and this function k are obtained. In this section, we illustrate the computation of latency distributions with a

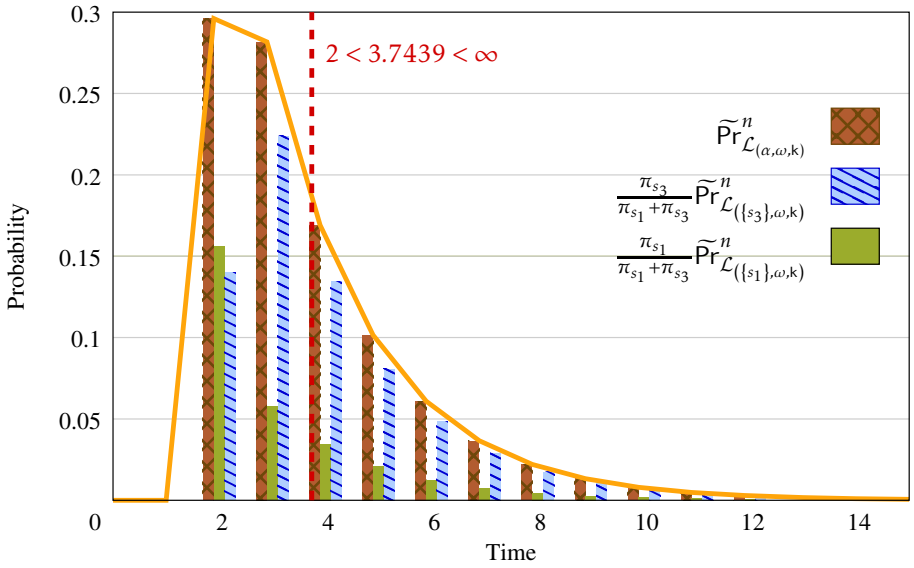


Figure 3.3: Latency distribution in $\mathcal{M}_P$

simple but significant example, and we present how α, ω, and k are fixed. To this aim, we use the DTAMC formalism: it provides us with on one side a DTMC and on the other side functional information allowing us to construct α, ω and k. We first recall that a latency is defined as the number of time-steps required from a state $s \in \alpha$ to observe k(s)-times a state of ω. In other words, an occurrence of latency starts in a state of $s \in \alpha$, and ends when the k(s)-th observation of a state of ω is processed.

The example chosen is the hardware FIFO queue presented in example 1.1 of chapter 1. The parameters of the queue are set as in example 2.1 of chapter 2. As a reminder, those parameters are:

- queue size Q_S is 1
- push operation delay D_{PUSH} is 2 time steps
- pop operation delay D_{POP} is 1 time steps
- production delay D_{PROD} is either 2 time steps with a probability 0.5 or 3 time steps with a probability 0.5
- consumption delay D_{CONS} is either 1 time step with a probability 0.5 or 4 time steps with a probability 0.5

This example is modeled by the DTAMC $\mathcal{M}_P^{\mathscr{A}} = \langle S, \rightsquigarrow, s_7, \mathscr{A} \rangle$ presented in example 2.12. The annotation function associates to each state four pieces of information:

- a boolean pushRq that is True if the state corresponds to the initiation of an element insertion in the queue (push request)
- a boolean popRq that is True if the state corresponds to the initiation of an element extraction from the queue (pop request)
- a boolean popRsp that is True if the state corresponds to the end of an element extraction from the queue (pop response),
- and an integer size that is the current size of the queue

The DTAMC $\mathcal{M}_P^{\mathscr{A}}$ is depicted in figure 2.8.

On this system, We are focusing on two different latencies:

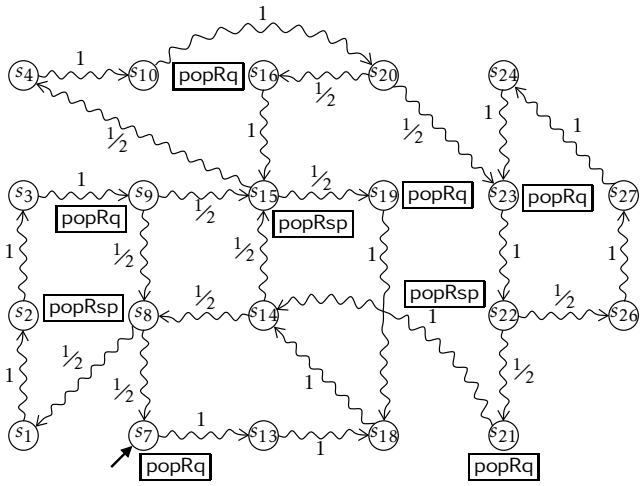


Figure 3.4: Minimized DTAMC of a 1-place FIFO queue DTMC model

- the pop operation latency
- the end-to-end latency

We use the methodology presented in previous sections to compute the distribution of those latencies. The end-to-end latency is an interesting notion that has already been studied [LMdS08] with other formalisms. In this chapter, we generalized the latency defined in [CHLS09] to be able to support end-to-end latencies.

3.4.1 Pop Operation Latency

The pop operation latency in the queue corresponds to the time elapsed between the pop request and the corresponding pop response. On the DTAMC model $\mathcal{M}_P^{\mathscr{A}}$, pop request and pop response actions are respectively identified by popRq and popRsp booleans associated to each state. This information defines the sets of states α and ω: α is the set of states where popRq is True, and ω is the set of states where popRsp is True.

The FIFO queue model ensures a behavior with no reentrant pop operations: a pop request will always be followed by a response before the next request. This information defines the function k: whatever the starting state $s \in \alpha$ of the latency, the first occurrence of a state in ω denotes the end of the latency, i.e., $\left(\forall s \in \alpha\right), \mathsf{k}(s) = 1$.

We can see that the only information used in the annotation function of $\mathcal{M}_P^{\mathscr{A}}$ to construct the sets α and ω and the function k are the booleans popRq and popRsp. We can thus consider the DTAMC $\mathcal{M}_P^{\mathscr{A}'} = \langle S, \rightsquigarrow, s_7, \mathscr{A}' \rangle$ presented in example 2.16, with annotation function $\mathscr{A}'$ limited to popRq and popRsp boolean values ($\mathscr{A}' : S \mapsto \{\mathsf{True}, \mathsf{False}\}^2$). Because $\mathcal{M}_P^{\mathscr{A}'}$ is not minimal, we compute the pop operation latency distribution on its quotient $\mathcal{M}_P^{\mathscr{A}'}{}_{/\approx}$ depicted in figure 2.10. The minimal DTAMC $\mathcal{M}_P^{\mathscr{A}'}{}_{/\approx}$ is drawn again in figure 3.4

The pop operation latency distribution, computed on $\mathcal{M}_P^{\mathscr{A}'}{}_{/\approx}$ is defined by:

- $\alpha = \{s_7, s_9, s_{16}, s_{19}, s_{21}, s_{23}\}$
- $\omega = \{s_8, s_{15}, s_{22}\}$
- $\mathsf{k} : \alpha \mapsto \mathbb{N}_+$

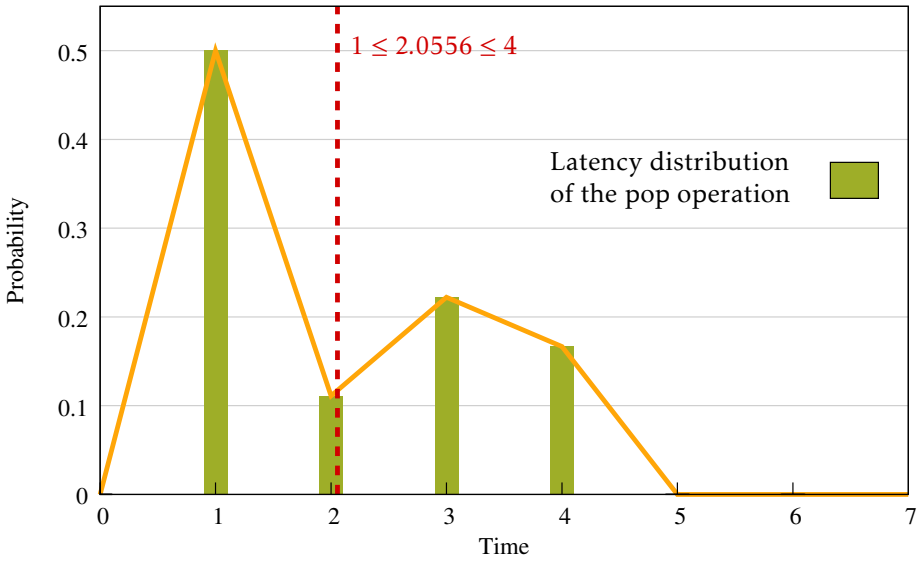


Figure 3.5: Distribution of the pop operation latency

$s \rightarrow 1$

The latency distribution of the pop operation for this system is depicted in figure 3.5. The latency definition provided in article [CHLS09] would have been sufficient to process this example.

3.4.2 Queue End-to-End Latency

The latency definition provided in article [CHLS09] is not sufficient when we consider latencies where several occurrences may overlap (the start of an occurrence of the latency may happen before the end of the previous one). This is typically the case of the end-to-end latency we study in this section.

The end-to-end latency in the queue corresponds to the time needed for an element to pass through the queue. More precisely, it is the time elapsed between the push request of an element and the corresponding pop response (which withdraws the same element). On the DTAMC model $\mathcal{M}_P^{\mathscr{A}}$, push request and pop response actions are respectively identified by pushRq and popRsp booleans associated to each state. This information defines the sets of states α and ω: α is the set of states where pushRq is True, and ω is the set of states where popRsp is True.

According to the number of elements in the queue, the first encountered pop response after a push request does not necessarily correspond to the withdrawal of the inserted element but may correspond to the withdrawal of a previously inserted element. Indeed, when a push request is processed, all the elements already present in the queue have to be withdrawn before the currently inserted one. This implies that there are a number of pop responses corresponding to the number of elements in the queue, to withdraw present elements. Then, the next pop response corresponds to the withdrawal of the element inserted. The function k depends consequently on the number of elements present in the queue when the push request is processed.

We can see that the only information used in the annotation function of $\mathcal{M}_P^{\mathscr{A}}$ to construct the

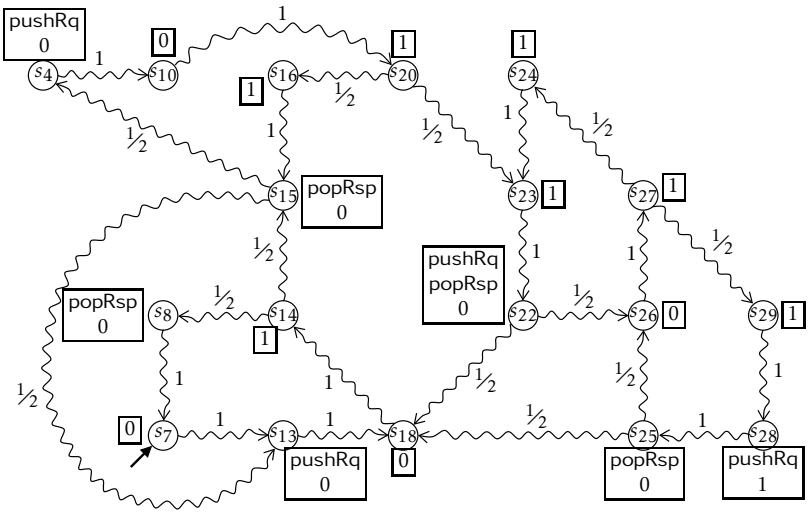


Figure 3.6: Minimized DTAMC of a 1-place FIFO queue DTMC model

sets α and ω and the function k are the booleans pushRq and popRsp and the integer size. We can thus consider the DTAMC $\mathcal{M}_P^{\mathscr{A}''} = \langle S, \rightsquigarrow, s_7, \mathscr{A}'' \rangle$, with annotation function $\mathscr{A}''$ limited to pushRq and popRsp boolean values and size integer value ($\mathscr{A}'' : S \mapsto \{\mathsf{True}, \mathsf{False}\}^2 \times \mathbb{N}$). $\mathcal{M}_P^{\mathscr{A}''}$ is consequently the DTAMC $\mathcal{M}_P^{\mathscr{A}}$ depicted in figure 2.8, ignoring the popRq boolean value. Looking at this figure, we can see that $\mathcal{M}_P^{\mathscr{A}''}$ is not minimal: $\{s_3, s_{18}, s_{21}\}$, $\{s_2, s_{13}, s_{19}\}$, $\{s_1, s_7\}$ and $\{s_9, s_{14}\}$ are equivalent classes with respect to $\sim_p$. We compute the pop operation latency distribution on its quotient $\mathcal{M}_P^{\mathscr{A}''}{}_{/\approx}$ depicted in figure 3.6.

The end-to-end latency distribution of the queue, computed on $\mathcal{M}_P^{\mathscr{A}''}{}_{/\approx}$ is defined by:

- $\alpha = \{s_4, s_{13}, s_{22}, s_{28}\}$
- $\omega = \{s_8, s_{15}, s_{22}, s_{25}\}$
- $\mathsf{k} : \alpha \mapsto \mathbb{N}_+$

$$s \rightarrow \begin{cases} 1 & \text{if} \quad s \in \{s_4, s_{13}, s_{22}\} \\ 2 & \text{if} \quad s = s_{28} \end{cases}$$

The end-to-end latency distribution for this system is depicted in figure 3.7.

3.4.3 Discussion

In this chapter we have introduced the latency performance measure on DTMCs. We provided a mathematical definition of the (long-run average) distribution of a latency and a way to compute it. The computation of the distribution relies on the calculation of steady state probabilities of the DTMC and transient analysis of extracted absorbing DTMCs. Actually, computing the latency distribution relies on the construction of the discrete phase-type distribution associated to the latency. We generalized the definition of a latency in a DTMC given in [CHLS09], which enables us to take into account end-to-end latencies.

The definition of a latency is sufficiently generic to cover a large amount of performance measures. Precisely, a latency may cover two of the three targeted performance measures we presented in introduction: throughput and latency. A throughput measure is generally the counterpart of a latency measure: it is the inverse of this latency measure. Hence, a throughput

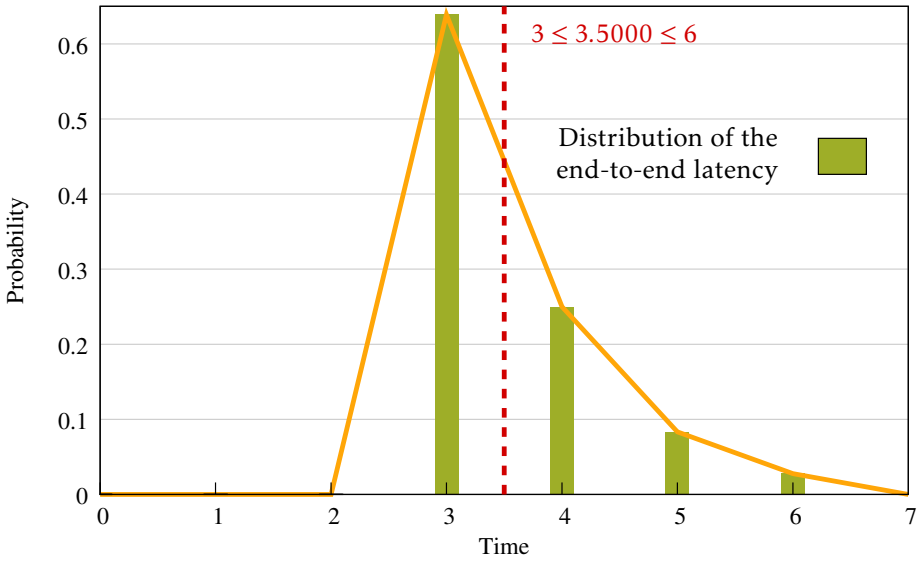


Figure 3.7: Distribution of the end-to-end latency

can be studied by using its associated latency. The latency in a system is directly linked to the definition of latency we gave in a DTMC: it is the amount of time elapsed between two particular time points that correspond to particular functional states of the system.

The definition of a latency in a DTMC relies on the identification of two sets of states. As illustrated in the examples, those two sets are identified according to a functional information, and annotated Markov chains are consequently a suitable formalism to associate functional information to states of a DTMC. As for every complex performance measure on Markov chains, a latency relies on our ability to construct a significant annotated Markov chain model for every kind of system.

Chapter 4

Interactive Markov Chains

As seen in chapters 2 and 3, Markov chains are an interesting formalism for performance evaluation. A lot of measures may be computed, when associating some functional information to states. Unfortunately, the Markov chain model of a large system is almost unachievable by direct means. Interactive Markov Chains (IMC) [Her02] are an elegant framework that combines LTS with CTMCs by strictly separating time and actions. The IMC formalism answers almost all the requirements we mentioned in introduction: models allow to tackle both functional verification and performance evaluation, they are constructed in a compositional way and the time model is compatible with delay of the systems to be modeled.

In IMCs, the underlying performance model is a CTMC. Consequently, IMCs provide an efficient way to construct accurate CTMC models of complex systems. IMCs could be an answer for the construction of annotated Markov chains, which would permit performance evaluation, as presented in chapter 2.

This chapter gives a short overview of the formalism of interactive Markov chains. We investigated how, using IMC models of complex hardware systems, performance measures could be computed. For those systems, some approximations are needed. Consequently, we identified the need of evaluating the accuracy of performance measures computed on underlying CTMCs of IMCs.

4.1 Interactive Markov Chains

4.1.1 Definition

Interactive Markov Chains are a stochastic extension of process algebra that strictly separates action-based transitions (also called interactive transitions) from time-based transitions (called Markovian transitions) representing delay whose duration follows an exponential distribution. Let $\mathcal{A}$ denote the set of actions, including an internal action τ. We assume that actions are taken instantaneously (but may be differed in time), and that Markovian transitions induce an exponentially distributed sojourn-time in states.

Definition 4.1 (Interactive Markov chain)**.** *An IMC is a tuple* $M = \langle S, \mathcal{A}, \longrightarrow, \dashrightarrow, s_0 \rangle$*, where*

- S *is a nonempty set of states,*
- $\mathcal{A}$ *is a finite set of actions including the internal action* τ*,*
- $\longrightarrow \subset S \times \mathcal{A} \times S$ *is a set of interactive transitions,*

– $\dashrightarrow \subset S \times \mathbb{R}_+ \times S \to \mathbb{N}$ *is a multiset of Markovian transitions,*
– *and* $s_0 \in S$ *is the initial state.*

$\mathscr{M}$ *will denote the set of all IMCs over* $\mathcal{A}$.

For convenience, we write $s_1 \xrightarrow{a} s_2$ rather than $(s_1, a, s_2) \in \longrightarrow$ and we write $s_1 \overset{\lambda}{\dashrightarrow} s_2$ rather than $(s_1, \lambda, s_2) \in \dashrightarrow$. In addition, $s_1 \longrightarrow s_2$ means that there exists an action $a \in \mathcal{A}$ such that $s_1 \xrightarrow{a} s_2$. Similarly, $s_1 \dashrightarrow s_2$ means that there exists a rate $\lambda \in \mathbb{N}_+$ such that $s_1 \overset{\lambda}{\dashrightarrow} s_2$.

To allow the composition of IMCs in a hierarchical and modular manner, we consider the language of IMCs, IMC_L (noted IML in [Her02]). IMC_L is defined by the following grammar. We use $\lambda, \mu, \nu, \ldots$ to range over rates (i.e., over $\mathbb{R}_+$), $a, a_1, \ldots$ to denote actions in $\mathcal{A}$, and $B, B_1, \ldots$ for behaviors denoting an expression of the language.

Definition 4.2 (Language IMC_L)**.** *Let A bet a set of actions not including* τ, $A \subset \mathcal{A} \setminus \{\tau\}$. *Behaviors of the language* IMC_L *are described by the grammar:*

$$B ::= \delta \mid a;B \mid \lambda;B \mid B_1 \,[\,]\, B_2 \mid B_1 \,|[A]|\, B_2 \mid \mathsf{hide}\ A\ \mathsf{in}\ B \mid \widetilde{B}$$

We write $\mathcal{B}$ to denote the set of all possible behaviors B. The operators of the language will be referred as:

– The termination symbol δ denotes a *blocked behavior*.
– The expression "$a;B$" denotes a sequential composition and is an *action-prefixed behavior*.
– The expression "$\lambda;B$" denotes also a sequential composition and is a *delay-prefixed behavior*.
– The expression "$B_1 \,[\,]\, B_2$" denotes a nondeterministic *choice* between B_1 and B_2.
– The expression "$B_1 \,|[A]|\, B_2$" denotes a *parallel composition* with a synchronization set (in LOTOS-style [BB87]).
– The expression "hide A in B" denotes abstraction and is an *hidden behavior*.
– Finally a possibly *recursive behavior* is defined by a rule of the form $\widetilde{B} = B$.

The formal semantics of the language IMC_L is defined in a structured operational semantics style by mapping each behavior B onto an IMC.

Definition 4.3 (Semantics of IMC_L behaviors)**.** *The operational semantics of a behavior* B_0 *over the set of actions* $\mathcal{A}$ *is defined as the IMC* $M = \langle \mathcal{B}, \mathcal{A}, \longrightarrow, \dashrightarrow, B_0 \rangle$, *where* $\longrightarrow$ *and* $\dashrightarrow$ *are defined by the inference rules of figure 4.1.*

We define the same operators as operators of IMC_L, with the same semantics for IMCs. For instance, "$M_1 \,|[A]|\, M_2$" denotes the parallel composition of two IMCs M_1 and M_2 with synchronization on actions of the set $A \in \mathcal{A}$.

The intuitive meaning of the semantics of IMC_L is the following:

– the blocked behavior δ cannot perform any action (including the internal τ action) and cannot interact anymore with its environment.
– The action-prefix "$a;B$" may perform the action a and then behaves like B (rule (1.a)).
– The delay-prefix "λ;B" will behave as B after a λ-exponentially distributed delay (rule (1.b)).
– The nondeterministic choice "$B_1 \,[\,]\, B_2$" depends on behaviors B_1 and B_2 (either they can take a Markovian transition or they can take an interactive transition). We can distinguish four possibilities:
  – Either behavior B_1 can take an interactive transition, $B_1 \xrightarrow{a} B_1'$. In this case, "$B_1 \,[\,]\, B_2$" can take a transition $\xrightarrow{a}$ and behaves like B_1' (rules (2.a)).

$\dfrac{}{a;B \xrightarrow{a} B}$	(1.a)	$\dfrac{}{\lambda;B \overset{\lambda}{\dashrightarrow} B}$	(1.b)
$\dfrac{B_1 \xrightarrow{a} B_1'}{B_1[\,]B_2 \xrightarrow{a} B_1'}$ $\dfrac{B_2 \xrightarrow{a} B_2'}{B_1[\,]B_2 \xrightarrow{a} B_2'}$	(2.a)	$\dfrac{B_1 \overset{\lambda}{\dashrightarrow} B_1'}{B_1[\,]B_2 \overset{\lambda}{\dashrightarrow} B_1'}$ $\dfrac{B_2 \overset{\lambda}{\dashrightarrow} B_2'}{B_1[\,]B_2 \overset{\lambda}{\dashrightarrow} B_2'}$	(2.b)
$\dfrac{B_1 \xrightarrow{a} B_1' \quad a \notin A}{B_1\|[A]\|B_2 \xrightarrow{a} B_1'\|[A]\|B_2}$ $\dfrac{B_2 \xrightarrow{a} B_2' \quad a \notin A}{B_1\|[A]\|B_2 \xrightarrow{a} B_1\|[A]\|B_2'}$ $\dfrac{B_1 \xrightarrow{a} B_1' \quad B_2 \xrightarrow{a} B_2' \quad a \in A}{B_1\|[A]\|B_2 \xrightarrow{a} B_1'\|[A]\|B_2'}$	(3.a)	$\dfrac{B_1 \overset{\lambda}{\dashrightarrow} B_1'}{B_1\|[A]\|B_2 \overset{\lambda}{\dashrightarrow} B_1'\|[A]\|B_2}$ $\dfrac{B_2 \overset{\lambda}{\dashrightarrow} B_2'}{B_1\|[A]\|B_2 \overset{\lambda}{\dashrightarrow} B_1\|[A]\|B_2'}$	(3.b)
$\dfrac{B_1 \xrightarrow{a} B_1' \quad a \notin A}{\mathsf{hide}\ A\ \mathsf{in}\ B_1 \xrightarrow{a} \mathsf{hide}\ A\ \mathsf{in}\ B_1'}$ $\dfrac{B_1 \xrightarrow{a} B_1' \quad a \in A}{\mathsf{hide}\ A\ \mathsf{in}\ B_1 \xrightarrow{\tau} \mathsf{hide}\ A\ \mathsf{in}\ B_1'}$	(4.a)	$\dfrac{B_1 \overset{\lambda}{\dashrightarrow} B_1'}{\mathsf{hide}\ A\ \mathsf{in}\ B_1 \overset{\lambda}{\dashrightarrow} \mathsf{hide}\ A\ \mathsf{in}\ B_1'}$	(4.b)
$\dfrac{\widetilde{B}=B \quad B \xrightarrow{a} B'}{\widetilde{B} \xrightarrow{a} B'}$	(5.a)	$\dfrac{\widetilde{B}=B \quad B \overset{\lambda}{\dashrightarrow} B'}{\widetilde{B} \overset{\lambda}{\dashrightarrow} B'}$	(5.b)

Figure 4.1: Operational semantics of $\mathsf{IMC}_{\mathsf{L}}$

  - Or behavior B_2 can take an interactive transition, $B_2 \xrightarrow{a'} B_2'$. In this case, "$B_1 \,[\,]\, B_2$" can take a transition $\xrightarrow{a'}$ and behaves like B_2' (rules (2.a)).
  - Or behavior B_1 can take a Markovian transition $B_1 \overset{\lambda}{\dashrightarrow} B_1'$. In this case "$B_1 \,[\,]\, B_2$" can take a transition $\overset{\lambda}{\dashrightarrow}$ and behaves like B_1' (rules (2.b)).
  - Or behavior B_2 can take a Markovian transition $B_2 \overset{\mu}{\dashrightarrow} B_2'$. In this case "$B_1 \,[\,]\, B_2$" can take a transition $\overset{\mu}{\dashrightarrow}$ and behaves like B_2' (rules (2.b)).
- The behavior "$B_1 \,|[A]|\, B_2$" depends also on behaviors B_1 and B_2, and we can distinguish five possibilities:
  - Either behavior B_1 can take an interactive transition, $B_1 \xrightarrow{a} B_1'$, with $a \notin A$. In this case, "$B_1 \,|[A]|\, B_2$" can take a transition $\xrightarrow{a}$ and behaves like "$B_1' \,|[A]|\, B_2$" (rules (3.a)).
  - Or behavior B_2 can take an interactive transition, $B_2 \xrightarrow{a'} B_2'$, with $a \notin A$. In this case, "$B_1 \,|[A]|\, B_2$" can take a transition $\xrightarrow{a'}$ and behaves like "$B_1 \,|[A]|\, B_2'$" (rules (3.a)).
  - Or behaviors B_1 and B_2 can take the same interactive transition, $B_1 \xrightarrow{a''} B_1'$ and $B_2 \xrightarrow{a''} B_2'$, with $a'' \in A$. In this case "$B_1 \,|[A]|\, B_2$" can take a transition $\xrightarrow{a''}$ and behaves like "$B_1' \,|[A]|\, B_2'$" (rules (3.a)).
  - Or behavior B_1 can take a Markovian transition $B_1 \overset{\lambda}{\dashrightarrow} B_1'$. In this case, "$B_1 \,|[A]|\, B_2$" can take a transition $\overset{\lambda}{\dashrightarrow}$ and behaves like "$B_1' \,|[A]|\, B_2$" (rules (3.b)).
  - Or behavior B_2 can take a Markovian transition $B_2 \overset{\mu}{\dashrightarrow} B_2'$. In this case, "$B_1 \,|[A]|\, B_2$" can take a transition $\overset{\mu}{\dashrightarrow}$ and behaves like "$B_1 \,|[A]|\, B_2'$" (rules (3.b)).
- The hidden behavior "hide A in B" behaves like B for which all actions of A are replaced by τ (rules (4.a) and (4.b)).
- Finally the behavior $\widetilde{B} = B$ behaves like B (rules (5.a) and (5.b)).

Notice that parallel composition is processed by interleaving of transitions (Markovian or interactive transitions) without adjusting rates of Markovian transitions. It is allowed by the memoryless property of the exponential distribution.

4.1.2 Properties of IMCs

We introduce several properties on IMCs. The first property deals with the competition between a τ-transition and Markovian transitions in a state of an IMC.

Definition 4.4 (Maximal progress)**.** *An IMC $M = \langle S, \mathcal{A}, \longrightarrow, \dashrightarrow, s_0 \rangle$ is said to be* maximal progress cut *if and only if*

$$\left(\forall (s,s') \in S^2\right) \quad s \xrightarrow{\tau} s' \Longrightarrow \left(\left(\nexists s'' \in S\right) \quad s \dashrightarrow s''\right)$$

Given an IMC M, we call "maximal progress cut IMC of M", written as $M_{/\!\!/_\tau}$, the largest maximal progress cut IMC contained in M.

The second property tackles the problem of competition between arbitrary interactive transitions (including τ), and Markovian transitions in an IMC.

Definition 4.5 (Urgency)**.** *An IMC* $M = \langle S, \mathcal{A}, \longrightarrow, \dashrightarrow, s_0 \rangle$ *is said to be* urgency cut *if and only if*

$$\left(\forall (s,s') \in S^2\right)\left(\forall a \in \mathcal{A}\right) \quad s \xrightarrow{a} s' \Longrightarrow \left(\left(\nexists s'' \in S\right) \quad s \dashrightarrow s''\right)$$

We can notice that an urgency cut IMC is by definition maximal progress cut. Given an IMC M, we call "urgency cut IMC of M", written as $M_{\not\dashrightarrow_{\mathcal{A}}}$, the largest urgency cut IMC contained in M.

Finally, we introduce a property linked to the number of interactive transitions that can be taken in a finite time interval.

Definition 4.6 (Non-Zeno)**.** *An IMC* $M = \langle S, \mathcal{A}, \longrightarrow, \dashrightarrow, s_0 \rangle$ *is said to be* non-Zeno *if and only if it can perform only finitely many actions in a finite time interval. We ensure this property on IMCs by bounding the number of possible actions taken in sequence (i.e., in a single time instant):*

$$\left(\forall (s,s') \in S^2\right)\left(\exists k \in \mathbb{N}\right)\left(\forall n \in \mathbb{N}\right)\left(\forall (s_1,\ldots,s_n) \in S^n\right)$$
$$s \longrightarrow s_1 \longrightarrow \cdots \longrightarrow s_n \longrightarrow s' \Longrightarrow (n \leq k)$$

The non-Zenoness property expresses that in a finite time interval, it is not possible to take an infinite number of interactive transitions. In particular, an IMC with loops of interactive transitions is not non-Zeno. The non-Zeno property allows to find a value k corresponding to the length of the longest sequence of interactive transitions in an IMC.

The definition of non-Zenoness property we gave is sometimes called *implementability* property [NS91]: the existence of a bound on the number of actions is ensured for any chosen time interval.

4.1.3 Markovian Strong and Branching Bisimulations

As seen for Markov chains in chapter 2, it is essential to have methods allowing to compare different processes. Comparing processes requires the definition of equivalence relations between processes. Bisimulations for IMCs, defined in [Her02], reflect the coexistence of interactive transitions and Markovian transitions. Bisimulations of IMCs rely on bisimulations for labeled transition systems [vGW96, Mil90] and bisimulations for CTMCs (a generalization for CTAMC was introduced in section 2.2.3).

The definition of bisimulations for IMCs are given considering an extension of the predicate γ_M introduced for CTAMCs in section 2.2.3. Consider an IMC $M = \langle S, \mathcal{A}, \longrightarrow, \dashrightarrow, s_0 \rangle \in \mathscr{M}$. The predicate $\gamma_M : S \times 2^S \mapsto [0,1]$ computes the cumulative transition rates from a state to a set of states,

$$\left(\forall s \in S\right)\left(\forall \mathcal{S} \subseteq S\right) \quad \gamma_M(s,\mathcal{S}) = \sum_{s' \in \mathcal{S}} \left\{\!\!\left\{ \lambda \mid s \xrightarrow{\lambda} s' \right\}\!\!\right\}$$

In addition,a new predicate,γ_0 for interactive transitions, is needed. The predicate $\gamma_0 : S \times \mathcal{A} \times 2^S \mapsto \{\mathsf{True}, \mathsf{False}\}$ states if, from a given state, it is possible to reach a set of states with a given action, i.e.,

$$\left(\forall s \in S\right)\left(\forall a \in \mathcal{A}\right)\left(\forall \mathcal{S} \subseteq S\right) \quad \gamma_0(s,a,\mathcal{S}) = \begin{cases} \mathsf{True} & \text{if } \left(\exists s' \in \mathcal{S}\right) \quad s \xrightarrow{a} s' \\ \mathsf{False} & \text{otherwise} \end{cases}$$

Finally, we introduce some notations. $s \overset{\tau}{\not\longrightarrow}$ is an abbreviation for $\neg\gamma_0(s,\tau,S)$ and means that s' is not able to fire a τ-transition. $\mathscr{M}/_{\mathscr{E}}$ denotes the set of equivalence classes of $\mathscr{M}$ with respect to relation $\mathscr{E}$ and $[s]_{/_{\mathscr{E}}}$ denotes the equivalence class with respect to $\mathscr{E}$ containing the state s.

Using those notations, one can provide the definition of the strong Markovian bisimulation that allows to compare IMCs. Intuitively, the strong Markovian bisimulation states that if two processes are strongly bisimilar, they have to simulate each other (functionally) and that for the overall transition rates of each process to the same equivalence class are equals.

Definition 4.7 (strong Markovian bisimulation of IMC)**.** *Strong Markovian bisimulation equivalence ($\sim$) is the coarsest equivalence relation on $\mathscr{M}$ such that $s_1 \sim s_2$ implies for all actions $a \in \mathcal{A}$ and all equivalence classes $\mathcal{C}$ of $\sim$ ($\mathcal{C} \in \mathscr{M}/_{\sim}$):*

(1) $\gamma_0(s_1,a,\mathcal{C}) \implies \gamma_0(s_2,a,\mathcal{C})$,

(2) $s_1 \overset{\tau}{\not\longrightarrow} \implies \left(s_2 \overset{\tau}{\not\longrightarrow} \wedge\ \gamma_M(s_1,\mathcal{C}) = \gamma_M(s_2,\mathcal{C})\right)$.

When comparing IMCs, one may want to abstract from internal computation, and thus internal transitions τ. The branching bisimulation [HL98, vGW96] is a weaker notion of equivalence that permits to abstract from internal computation.

Definition 4.8 (IMC branching Markovian bisimulation)**.** *Branching Markovian bisimulation equivalence ($\approx$) is the coarsest equivalence relation on $\mathscr{M}$ such that $s_1 \approx s_2$ implies for all action $a \in \mathcal{A}$ and all equivalent class $\mathcal{C}$ of $\approx$ ($\mathcal{C} \in \mathscr{M}/_{\approx}$):*

(1) $\gamma_0(s_1,a,\mathcal{C}) \implies$ *either* $(a = \tau \wedge s_2 \in \mathcal{C})$ *or* $\left(\exists s_2'\right)\left(s_2 \overset{\tau*}{\longrightarrow} s_2' \wedge s_1 \approx s_2' \wedge \gamma_0(s_2',a,\mathcal{C})\right)$

(2) $s_1 \overset{\tau}{\not\longrightarrow} \implies \left(\exists s_2' \overset{\tau}{\not\longrightarrow}\right)\left(s_2 \overset{\tau*}{\longrightarrow} s_2' \wedge s_1 \approx s_2' \wedge \gamma_M(s_1,\mathcal{C}) = \gamma_M(s_2',\mathcal{C})\right)$

Two IMCs M_1 and M_2 are said to be strong (resp. branching) Markovian bisimilar, if their initial states are strong (resp. branching) Markovian bisimilar.

Lemma 4.1. *If two IMCs are strong Markovian bisimilar, they are also branching Markovian bisimilar:*

$$\left(\forall (M_1,M_2) \in \mathscr{M}^2\right) \quad M_1 \sim M_2 \Longrightarrow M_1 \approx M_2$$

Proof. Consider two states s_1 and s_2 strongly Markovian bisimilar, $s_1 \sim s_2$, and an equivalence class $\mathcal{C} \in \mathscr{M}/_{\sim}$. By definition, if $\gamma_0(s_1,a,\mathcal{C})$ holds, then $\gamma_0(s_2,a,\mathcal{C})$ holds. Taking $s_2' = s_2$, condition *(1)* of the branching Markovian bisimulation is verified.

On the same idea, if $s_1 \overset{\tau}{\not\longrightarrow}$, then $s_2 \overset{\tau}{\not\longrightarrow}$ and $\gamma_P(s_2,\mathcal{C}) = \gamma_P(s_1,\mathcal{C})$. Taking $s_2' = s_2$, condition *(2)* of the branching Markovian bisimulation is verified. □

In addition to comparing IMCs, a bisimulation can be used to define the smallest IMC (in terms of state space size) equivalent to another one. This smallest equivalent IMC is called *quotient*. One may define a quotient according to the strong Markovian bisimulation $\sim$ and a quotient according to the branching Markovian bisimulation $\approx$.

Definition 4.9 (IMC quotient)**.** *Given an IMC $M = \langle S, \mathcal{A}, \longrightarrow, \dashrightarrow, s_0\rangle$ and a bisimulation $\mathscr{E}$, the quotient of M according to $\mathscr{E}$ is the IMC $M_{/_{\mathscr{E}}} = \langle S_{/_{\mathscr{E}}}, \mathcal{A}, \longrightarrow_{/_{\mathscr{E}}}, \dashrightarrow_{/_{\mathscr{E}}}, [s_0]_{/_{\mathscr{E}}}\rangle$. $M_{/_{\mathscr{E}}}$ verifies:*

– *$M_{/_{\mathscr{E}}}$ and M are bisimilar ($M_{/_{\mathscr{E}}}\ \mathscr{E}\ M$)*

– *for every IMC M', defined over the state space S', and bisimilar to M (and to $M_{/\mathscr{E}}$), $M' \mathscr{E} M$, we have* $\mathsf{card}(S_{/\mathscr{E}}) \leq \mathsf{card}(S')$

We say that an IMC is minimal (with respect to a bisimulation $\mathscr{E}$) if its size is the same as the size of its quotient.

In the previous section, we introduced maximal progress cut and urgency cut allowing to define subclasses of IMCs. The maximal progress cut present an interesting property according to the bisimulations: maximal progress cut preserves the strong Markovian bisimulation.

Lemma 4.2. *An IMC M and its maximal progress cut IMC $M_{\not\to_\tau}$ are strongly Markovian bisimilar:*

$$\left(\forall M \in \mathscr{M}\right) \quad M \sim M_{\not\to_\tau}$$

Proof. The condition *(2)* of the strong Markovian bisimulation is only defined for states that do not allow to take a τ-transition. Because a maximal progress cut IMC $M_{\not\to_\tau}$ only differs from M on states allowing to take a τ-transition, this property is ensured. □

Contrary to the maximal progress, the urgency cut does not preserve bisimulations. Indeed by cutting Markovian transitions that compete with interactive ones, the condition *(2)* of strong and branching Markovian bisimulations are not ensured.

As corollary, lemma 4.2, together with lemma 4.1, implies that maximal progress cut preserves the branching Markovian equivalence:

Lemma 4.3. *An IMC M and its maximal progress cut IMC $M_{\not\to_\tau}$ are branching Markovian equivalent:*

$$\left(\forall M \in \mathscr{M}\right) \quad M \approx M_{\not\to_\tau}$$

Proof. Direct implication from lemma 4.2 and lemma 4.1. □

Finally, we can deduce that the quotient of an IMC with respect to the branching Markovian bisimulation is also maximal progress cut.

Lemma 4.4. *The quotient $M_{/\approx}$ of an IMC M is maximal progress cut:*

$$\left(\forall M \in \mathscr{M}\right) \quad \left(M_{/\approx}\right)_{\not\to_\tau}$$

Proof. By definition, for every IMC M, the state space of $M_{\not\to_\tau}$ is smaller or equal to the state space of M. In particular, given an IMC M, the state space of $\left(M_{/\approx}\right)_{\not\to_\tau}$ is smaller or equal to the state space of $M_{/\approx}$.

The definition of a quotient (definition 4.9) ensures that for every IMC M' branching Markovian bisimilar to $M_{/\approx}$, the state space of $M_{/\approx}$ is smaller or equal to the state space of M'. Because $\left(M_{/\approx}\right)_{\not\to_\tau} \approx M_{/\approx}$ (lemma 4.3) we have that $\left(M_{/\approx}\right)_{\not\to_\tau}$ and $M_{/\approx}$ have the same state space: the quotient is consequently maximal progress cut. □

4.1.4 Congruence Property of Markovian Branching Bisimulation

One may be interested in the preservation of the bisimulations with respect to the composition operators of $\mathbb{IMC}_L$. Indeed, it is interesting to know if applying a same operator (sequence, composition, etc.) of $\mathbb{IMC}_L$ on two equivalent IMCs yields two equivalent IMCs. This so-called *congruence* property is of importance for the parallel composition of IMCs ($|[\cdots]|$). Indeed, the state space explosion may mainly occur when composing processes in parallel. In this case, it is interesting to reduce state space of processes before composing them.

Theorem 4.1 (Congruence with respect to $|[\cdots]|$)**.** *The branching Markovian bisimulation is a congruence with respect to the parallel composition operator, i.e.,*

$$\left(\forall (M_1, M_2) \in \mathscr{M}^2\right)\left(\forall M_3 \in \mathscr{M}\right)\left(\forall A \subseteq \mathcal{A}\backslash\{\tau\}\right) \quad M_1 \approx M_2 \Longrightarrow M_1 \,|[A]|\, M_3 \approx M_2 \,|[A]|\, M_3$$

The congruence property with respect to composition operators in IMCs authorizes to follow a compositional approach: models can be minimized at intermediate composition to reduce the state space. It is a solution to circumvent the state space explosion problem.

4.2 Interactive Markov Chains in Practice

In this section, we briefly present a methodology that can be applied to model complex systems using IMCs. Then, we investigate the accuracy of performance measures obtained from IMCs, in particular when using IMCs to model constant delays.

4.2.1 Modeling and Analysis of Systems with Interactive Markov Chains

In this section, we focus on the methodology to be followed to model and analyze systems using the IMC formalism.

Modeling a hardware system with IMCs consists in writing models with time information expressed as exponentially distributed delays. Arbitrary continuous delay distributions can be fit arbitrarily close by continuous phase-type distributions (characterized by absorbing CTMC, see chapter 2), which provides us an interesting time model for IMCs, for which any kind of delays can be taken into account. Consequently, the simplest solution to model systems with IMCs consists in introducing delays compositionally, in a constraint-oriented way: delays, modeled by continuous phase-type distributions in separated processes, are seen as time constraints and composed in parallel with the functional specification of the system to be modeled.

The modeling of arbitrary delays by continuous phase-type distributions is at the heart of the modeling methodology of IMCs. As a consequence, some details concerning the way to model arbitrary delays are needed. We can first analyze the different kind of delays we want to model. One can distinguish two types of delays:

– **Definite delays**. Those delays are known by designers and represent a characteristic of the system. In hardware systems, they correspond to time characteristic of hardware elements and they typically consist in a constant amount of time (counted in a number of clock cycles).

- **Delay patterns**. Those delays are not precisely known and are defined according to assumptions concerning a timed behavior. One can distinguish two different use of this kind of delays:
  - Either they are out of control of designers and are independent of the system. For instance, it is usual to suppose that the expected life time of a component is exponentially distributed.
  - Or they may be used to abstract from a real behavior that is not precisely known. As an example, we will use delay patterns to model applications. Since application behaviors depend on implementation (and may vary according to external influence), we will use delay patterns to model them, assuming that the chosen pattern is a good representation of a real behavior.

The approximation of an arbitrary delay by a continuous phase-type distribution can be made as close as wanted from the distribution [Neu81]. In most cases, obtaining a better approximation of a delay consists in considering a phase-type distribution characterized by an absorbing CTMC with a larger state space. Efficient tools [HT02] implement the phase-type approximation of arbitrary distributions, the quality of the approximation being given by limiting the state space of the resulting CTMC.

Hence the best phase-type approximation may be characterized by a CTMC with an infinite state space. Because we are targeting finite-state space models, one has to find a trade-off between the accuracy of the phase-type approximation and the size of the state space of the corresponding absorbing CTMC. We consequently distinguish two kind of phase-type models:

- **Exact models**. If the delay to be modeled still follows a continuous phase-type distribution (for instance an exponentially distributed delay), the model is exact in the sense that there is no error between the model and the delay to be modeled.
- **Approximated models**. If the delay to be modeled does not follow a continuous phase-type distribution, the model is approximated in the sense that we limit the state space of the characterizing CTMC, inducing an error between the model and the delay to be modeled.

Delays are consequently exactly or approximately modeled by continuous phase-type distributions and can be inserted in the functional model of the system. After inserting delays, we obtain an IMC that can be analyzed to extract performance results. The analysis of an IMC relies on an underlying model that is a Markov decision process (MDP), i.e., a CTMC presenting nondeterminism. In most cases, and under some conditions (for instance, if there is no under-specification in the model), the underlying model is simply a CTMC, which can be obtained using properties like maximal progress, urgency cut, or applying branching minimization. A description of the problem of nondeterminism and under-specification in IMCs can be found in [Her02]. In the following, we consider that, from an IMC, we are able to generate a CTMC to be analyzed.

4.2.2 Accuracy of CTMC Performance Analysis

The performance measures we target (latency, throughput and resource utilization) rely on state probabilities of the underlying CTMC of an IMC (mainly steady state probabilities). Consequently, the accuracy of those measures depends a lot on the accuracy of state probabilities. In a first approach we can identify two sources of errors concerning the state probabilities.

Firstly, numerical computations may induce errors. This problem is mainly related to the

implementation of the algorithms used to obtain state probabilities. We do not tackle the problem of accuracy related to those errors since it is out of the scope of this thesis.

Secondly, phase-type approximations of arbitrary delay distributions may induce errors. To avoid the state space explosion problem, if delays to be modeled do not follow a continuous phase-type distribution, their approximated phase-type model must have a state space as small as possible. A trade-off between the accuracy of the results (i.e., accuracy of state probabilities) and the state space of the model has to be found. To reach this trade-off, a possible solution is the study of a confidence interval on computed performance results, as a function of the individual errors relative to each phase-type approximation.

Concerning errors due to phase-type approximations of arbitrary delay distributions, one has to differentiate delay patterns from definite delays. For a delay pattern, we are not able to evaluate the error between the phase-type model and the delay, because we do not know the real delay to be modeled but just an assumed pattern. In this case, we do not consider that the phase-type model induces an error on the computed results. The computed performance results are thus given under the hypothesis that the phase-type model used is a reasonable assumption, representative of the real timed behavior.

In the rest of this section, we illustrate the difficulty of computing a confidence interval for performance measures computed from IMCs.

4.2.3 Bounding the Error on Performance Measures: Hardware Constant Delays Example

Theoretically, it is possible to approximate every arbitrary distributed delay with an error tending to zero. If the delay to be modeled does not follow a phase-type distribution, the state space of the absorbing CTMC characterizing its phase-type model may tend to infinity when the error tends to zero. We illustrate this fact by considering the phase-type approximation of a constant delay.

Example 4.1. *An Erlang distribution is a phase-type distribution characterized by two parameters k and λ. It is a sequence of k successive exponential distributions (k phases) characterized by the parameter λ. The cumulative distribution function and expected value of a random variable X, following an Erlang distribution $erl(k, \lambda)$, are*

$$F_{k,\lambda}(x) = \Pr\{X \leq x\} = 1 - \sum_{n=0}^{k-1} \frac{\exp^{-\lambda x}(\lambda x)^n}{n!} \quad \textit{and} \quad E[X] = \frac{k}{\lambda}$$

As all phase-type distributions, it is characterized by an absorbing CTMC

$$Q = \begin{bmatrix} -\lambda & \lambda & 0 & 0 \\ 0 & \ddots & \ddots & 0 \\ 0 & 0 & -\lambda & \lambda \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad \textit{and} \quad \pi(0) = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

Consider a constant delay. It is an important delay to model in hardware systems (hardware operations take a constant amount of time). Erlang distributions are good candidates to approximate constant delays. Consider the constant delay with duration D for which the cumulative distribution

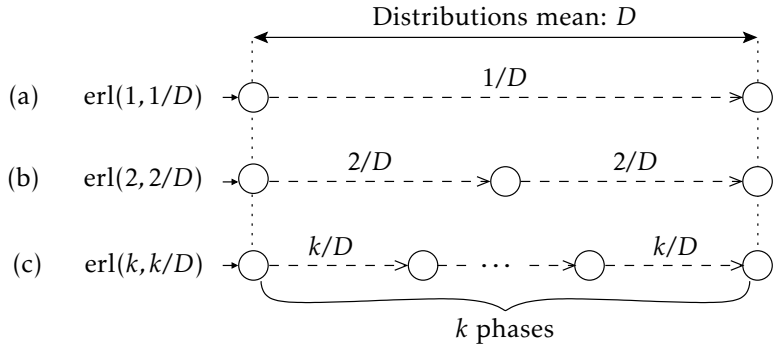


Figure 4.2: Different Erlang distributions with identical mean values

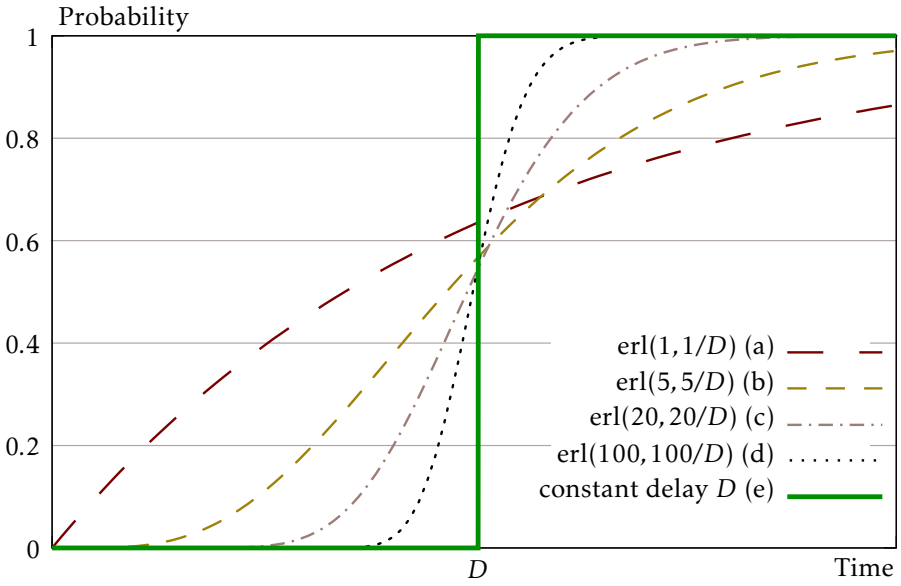


Figure 4.3: Approximation of a constant delay D by different Erlang distributions

is depicted by the solid line (e) in figure 4.3. The simplest approximation, adjusting its mean to D, is the erl$(1, 1/D)$ *distribution or exp*$(1/D)$ *(figure 4.2(a) and line (a) in figure 4.3). When increasing the number of phases, still keeping the overall mean to D (i.e., when the number of phases increases, we decrease rates to keep the same overall mean value D, as illustrated in figure 4.2(c)), the Erlang distribution is closer to the constant delay D as depicted in figure 4.3. Theoretically, the best approximation of a constant delay is an Erlang distribution with an infinite number of phases* ($k \to \infty$) *and* λ *tending to zero.*

In practice, because we are dealing with models that have to present a finite state space, the number of phases of the phase-type models must be limited. The limitation of phase-type models is needed if we want to avoid the state-space explosion problem, when composing models in parallel: the more parallelism there is, The more state-spaces of phase-type models have to be reduced to avoid the state-space explosion. Indeed, when considering parallelism for phase-type modeled delays, state spaces of modeled systems increase exponentially with the number of phases of the approximations.

Example 4.2. *The state space of the IMC corresponding to the parallel composition of n k-phases Erlang distributions has k^n states.*

Since we are not able to deal with infinite state space phase-type and infinite CTMC models, a trade-off between the accuracy of the targeted performance measure and the model size has to be found. This kind of trade-off is usual: measures are based on initial hypotheses (our approximations), and, according to the final confidence interval on the measures, hypotheses are adjusted to minimize the error. Unfortunately, a drawback of this methodology is that it assumes the margin of error on the computed measure to be available.

The first task in the computation of a confidence interval on performance measures is the approximation of the error between each real delay to be modeled and its phase-type approximation. The distribution of this error gives us an information on how the model diverges from the real delay.

Example 4.3. *Consider an Erlang distribution erl$(k,k/D)$, with cumulative distribution function $F(t)$, approximating a constant delay D, with cumulative distribution function $C(t)$. We define the error between the two distribution functions as the function $e(t) = |C(t) - F(t)|$. We are interested in the cumulative distribution function $E(t)$ of the error $e(t)$. This example is depicted in figure 4.4.*

The cumulative distribution function $E(x)$ of the error is given by

$$E(x) = \mathsf{Pr}\{e(t) \leq x\} = \begin{cases} 1 & \text{if } x \geq f(D) \\ E_p/E_m(x) & \text{if } x < f(D) \end{cases}$$

with E_m the area under $e(t)$ depicted as a filled area in figure 4.4 (formally the integral of $e(t)$ between 0 and $+\infty$), and $E_p(x)$ the area under $e(t)$ for t taking values between 0 and $f^{-1}(x)$ and between $1 - f^{-1}(x)$ and $+\infty$ depicted by the hashed area in figure 4.4 for $x = a$.

For instance, with the knowledge of $E(x)$, it is possible to adjust the number of phases of the Erlang distribution to provide results like "the error between the constant delay D and the Erlang approximation is less than $D/10$ with a probability 0.95".

Then, knowing all the errors between definite delays and their respective phase type approximations, a new question arises: how do those errors combine with respect to the computed steady state probabilities in the CTMC, and is it possible to get the margin of error on the targeted performance measure? This question has no simple answer, errors due to approximations may compensate each other, implying an exact result on steady state probabilities, or they may cumulate, inducing an error on the steady state probabilities that is not obviously linked to approximation errors. We illustrate this problem on the following example.

Example 4.4. *We illustrate the problem of evaluation of the error on steady state probabilities of the CTMC with respect to the errors of phase type approximations. Consider three different operations O_1, O_2 and O_3, characterized by execution times D_1, D_2 and D_3, which are definite delays. We focus on three different configurations, which are simple illustrations of typical behaviors allowed by process algebras:*

- *Operations O_1 and O_2 are executed in sequence. At the end of the operation O_2, the operation O_3 is processed. This configuration consists in executing the operation O_3 after the sequence of operations O_1 and O_2, i.e., after a delay equal to $D_1 + D_2$.*
- *Operations O_1 and O_2 are started simultaneously. When the first operation of O_1 and O_2 ends, the operation O_3 is processed. This configuration corresponds to parallelism of operations O_1*

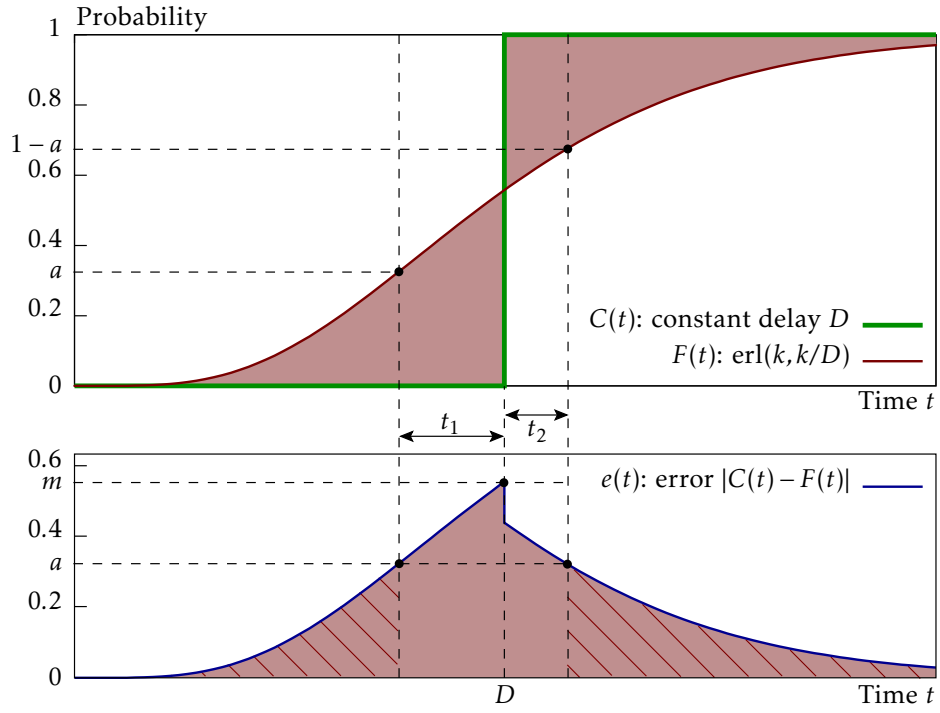


Figure 4.4: Approximation of a constant delay D by different Erlang distributions

and O_2 with race condition: the first operation ending enables the start of O_3. Consequently, O_3 is started after a delay equal to $\min(D_1, D_2)$.

– *Operations O_1 and O_2 are started simultaneously. When both operations O_1 and O_2 are processed, the operation O_3 is processed. This configuration corresponds to the parallelism of operations O_1 and O_2 with synchronization point at the end of the operations, enabling the execution of operation O_3. Consequently, O_3 is started after a delay equal to* $\max(D_1, D_2)$.

In the three configurations, at the end of the operation O_3, the system loops. We consider that delays D_1, D_2 are constants, $D_1 = 4$ time units and $D_2 = 3$ time units, and that delay D_3 is exponentially distributed, on average equal to 2 time units. We study the long-run average fraction of time, Π_{O_3}, spent to process O_3.

Because those configurations are simple, one can first compute the exact solution of the problem. For instance, semi-Markov chain formalism [LHK01] can be used to compute exact results:

– *first configuration:* $\Pi_{O_3} = \frac{D_3}{D_1+D_2+D_3} = \frac{2}{9}$

– *second configuration:* $\Pi_{O_3} = \frac{D_3}{\min(D_1,D_2)+D_3} = \frac{2}{5}$

– *third configuration:* $\Pi_{O_3} = \frac{D_3}{\max(D_1,D_2)+D_3} = \frac{1}{3}$

Secondly, one can compare the exact results with approximated results obtained using phase-type approximations of delays. The constant delays D_1, D_2 are approximated by Erlang distributions $\widetilde{D_1}$, $\widetilde{D_2}$:

– *$\widetilde{D_1}$ is an Erlang distribution given by parameters (n_1, λ), with $\lambda = \frac{n_1}{D_1}$*

– *$\widetilde{D_2}$ is an Erlang distribution given by parameters (n_2, μ), with $\mu = \frac{n_2}{D_2}$.*

For each configuration, one can study the long-run average fraction of time Π_{O_3} to process operation O_3, with respect to the quality of Erlang approximations (i.e., the number of phases used) of delays D_1 and D_2. Actually, for those configurations, the error on the long-run average fraction

of time spent in each state is directly linked to the errors between the expected value of sequence, minimum and maximum of Erlang distributions, with respect to the values of sequence, minimum and maximum of the real delays Erlang distributions have to model.

For the time duration of the sequence of operations O_1 and O_2, the considered model relies on the sequence of the Erlang distributions $\widetilde{D_1}$ and $\widetilde{D_2}$. Whatever is the length of the Erlang distributions $\widetilde{D_1}$ and $\widetilde{D_2}$ (i.e., for every values n_1 and n_2), the expected value of $\widetilde{D_1} + \widetilde{D_2}$ is equal to $D_1 + D_2$:

$$E\left[\widetilde{D_1} + \widetilde{D_2}\right] = E\left[\widetilde{D_1}\right] + E\left[\widetilde{D_2}\right] = D_1 + D_2$$

In this case, on average, there is no error induced by the Erlang approximations. Consequently, there is no error on Π_{O_3} between the real system and the model with Erlang approximations.

For the time duration of the minimum of operations O_1 and O_2, the absorbing CTMC characterizing the minimum of the Erlang distributions $\widetilde{D_1}$ and $\widetilde{D_2}$ is depicted in figure 4.5. We can study the expected value of the minimum of $\widetilde{D_1}$ and $\widetilde{D_2}$ according to the number of phases n_1 and n_2. We depict this minimum for n_1 and n_2 ranging between 1 and 30 in figure 4.6. Notice that the expected value of the minimum returned by the model is always under-approximated (equal to 1.71 time units in the case of using two exponential distributions to approximate delays D_1 and D_2, against a real value of 3). The long-run average fraction of time spent to process O_3 is consequently always over-evaluated.

For the time duration of the maximum of operations O_1 and O_2, the absorbing CTMC characterizing the maximum of the Erlang distributions $\widetilde{D_1}$ and $\widetilde{D_2}$ is depicted in figure 4.7. We can study the expected value of the maximum of $\widetilde{D_1}$ and $\widetilde{D_2}$ according to the number of phases n_1 and n2. We depict this maximum for n_1 and n_2 ranging between 1 and 30 in figure 4.8. Notice that the expected value of the maximum returned by the model is always over-approximated (equal to 5.29 time units in the case of using two exponential distributions to approximate delays D_1 and D_2, against a real value of 4). The long-run average fraction of time spent to process 0_3 is consequently always under-evaluated.

Moreover, considering a different value for D_3 has an impact on the long-run average time spent to process O_3. Indeed, the relative weight of operation O_3 on the time execution of the system is modified.

The previous example illustrates that, errors due to phase-type approximations may be counterbalanced or may induce an error on the steady-state probabilities of the system. Even considering very simple systems, it seems to be intractable to predicate how phase-type approximation errors of definite delays combine, with respect to the targeted performance measures.

4.3 Discussion

In this chapter, we presented interactive Markov chains, a formalism that can theoretically provide an answer to almost all problems concerning performance evaluation of hardware systems. Indeed, IMCs can be used to compositionally model complex hardware systems with a great precision and to generate CTMCs that would be unachievable by direct means.

Theoretically, IMCs authorizes to model any kind of timed system with an arbitrary accuracy. Nevertheless, for some delays to be modeled, the counterpart of an arbitrary accuracy is that infinite or very large state-space models are required. This is mainly the case for constant delays widely present in hardware systems. This is not compatible with our aim of exploring

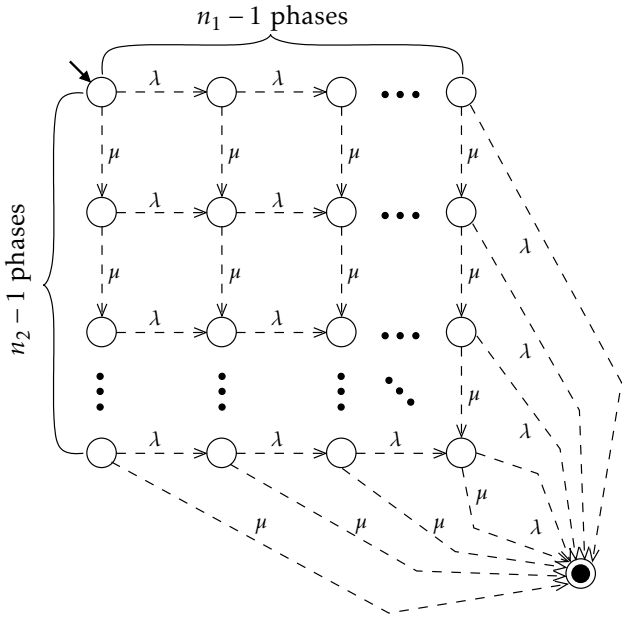


Figure 4.5: Phase type model of the minimum of two Erlang distributions with parameters (λ,n_1) and (μ,n_2)

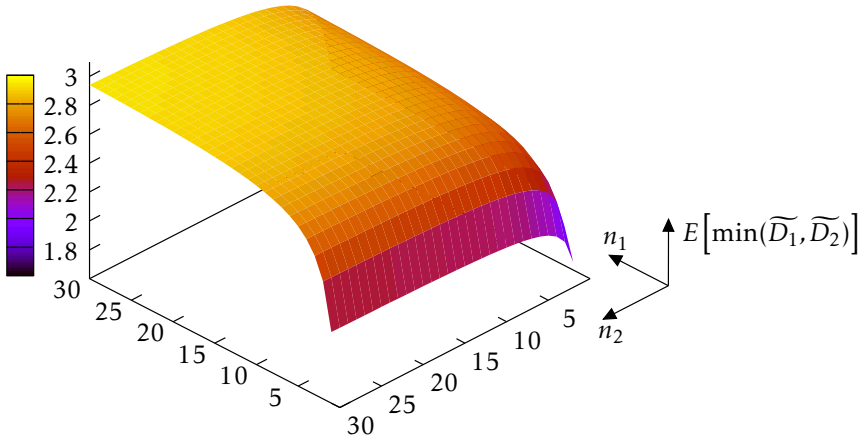


Figure 4.6: Expected value of the minimum of two Erlang distributions $\widetilde{D_1}$ and $\widetilde{D_2}$ with parameters $(\lambda = \frac{n_1}{4},n_1)$ and $(\mu = \frac{n_2}{3},n_2)$

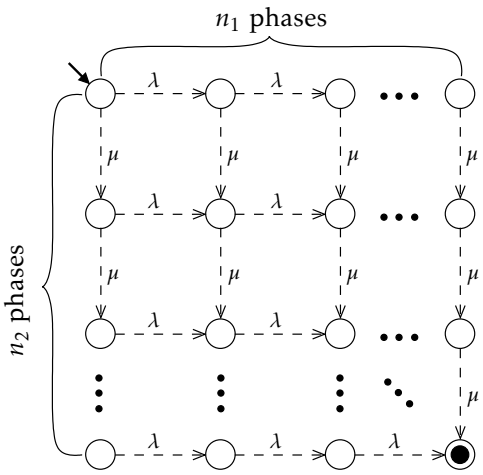


Figure 4.7: Phase type model of the maximum of two Erlang distributions with parameters (λ,n_1) and (μ,n_2)

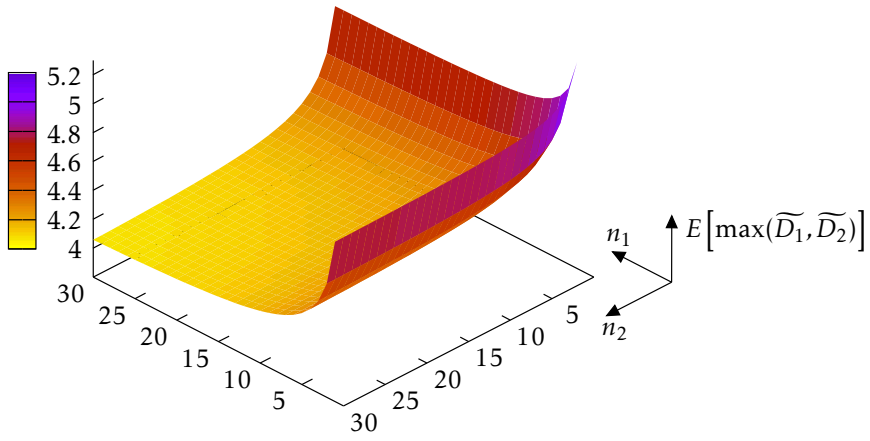


Figure 4.8: Expected value of the maximum of two Erlang distributions $\widetilde{D_1}$ and $\widetilde{D_2}$ with parameters ($\lambda = \frac{n_1}{4}$,n_1) and ($\mu = \frac{n_2}{3}$,n_2)

of the entire state-space of models. A trade-off between the state space of models and their accuracy has to be reached.

As a consequence, approximations of delays are mandatory to limit the size of the state space and to process a numerical analysis of models. Approximations deliberately introduce errors in a model. Thus, to trust performance results computed on IMCs, an evaluation of the impact of introduced errors on computed performance results is required. However, we are not aware of a way to evaluate the impact of approximation errors on the performance results. Actually, we showed that depending on the interaction of the models we compose, approximation errors may be additive or destructive.

Another approach would be to use another formalism, which would allow to model delays precisely and thus avoid to introduce approximation errors. We present such a model in the next chapter by switching to a discrete-time context, adequate for modeling constant delays of hardware systems.

Chapter 5

Interactive Probabilistic Chains

This chapter introduces the formalism of Interactive Probabilistic Chains (IPC), which combines labeled transition systems and discrete-time Markov chains, transposing interactive Markov chains [Her02] in a discrete-time context. Although IMCs and IPCs share similarities, mainly in the way they provide compositionality to scale to models of large systems, the different nature of time between IMCs and IPCs (continuous versus discrete) implies significant differences in their construction and analysis. The IPC formalism aims at avoiding errors due to approximation of constant delays (as in IMCs), whilst preserving its strong points: hierarchical composition and abstraction abilities that are mandatory for an industrial use.

In a first section, we define the IPC formalism. Then, we define a way to associate a DTAMC and an SMC to an IPC so as to allow performance evaluation. In a third section, we illustrate the use of the IPC formalism on two examples. We show how the associated DTAMC is constructed in practice, to ensure the applicability of performance evaluation methods presented in previous chapters (such as latency distribution computation). Finally, we study the relation between IMC and IPC formalisms.

5.1 Interactive Probabilistic Chains

In this section we present the syntax and operational semantics of IPCs. We introduce several properties, used in the next sections, which define subclasses of IPCs. We also discuss strong and branching bisimulations of IPCs, allowing to compare and minimize them.

5.1.1 Definition

An IPC has to be expressive enough to depict timed probabilistic transitions and interactive transitions. Timed probabilistic transitions (simply called probabilistic transitions) refer to DTMC transitions considering each probabilistic transition as a time step. Interactive transitions refer to transitions of labeled transition systems and are based on atomic actions. Let $\mathcal{A}$ denote the set of actions, including an internal action τ. We assume that actions are taken instantaneously (but may be differed in time), and that every probabilistic choice takes exactly one time step.

Definition 5.1 (Interactive probabilistic chain)**.** *An IPC is a tuple* $P = \langle S, \mathcal{A}, \longrightarrow, \rightsquigarrow, s_0 \rangle$, *where*

– S *is a nonempty set of states,*
– $\mathcal{A}$ *is a finite set of actions including the internal action* τ*,*
– $\longrightarrow \subset S \times \mathcal{A} \times S$ *is a set of interactive transitions,*
– $\rightsquigarrow \subset S \times]0,1] \times S \to \mathbb{N}$ *is a multi-set of probabilistic transitions, satisfying:*

$$\Big(\forall s \in S\Big) \qquad \Big(\big(\exists s' \in S\big)\big(\exists p \in]0,1]\big) \quad s \overset{p}{\rightsquigarrow} s'\Big) \Longrightarrow \sum_{s'} \Big\{\!\!\Big\{ p \mid s \overset{p}{\rightsquigarrow} s' \Big\}\!\!\Big\} = 1$$

– *and* $s_0 \in S$ *is the initial state.*

$\mathscr{P}$ *will denote the set of all IPCs over* $\mathcal{A}$*.*

For convenience, we write $s_1 \overset{a}{\longrightarrow} s_2$ rather than $(s_1, a, s_2) \in \longrightarrow$ and $s_1 \overset{p}{\rightsquigarrow} s_2$ rather than $(s_1, p, s_2) \in \rightsquigarrow$. In addition, $s_1 \longrightarrow s_2$ means that there exists an action $a \in \mathcal{A}$ such that $s_1 \overset{a}{\longrightarrow} s_2$. Similarly, $s_1 \rightsquigarrow s_2$ means that there exists a probability $p \in]0,1]$ such that $s_1 \overset{p}{\rightsquigarrow} s_2$.

To allow the use of IPCs in a hierarchical and modular manner, we have to discuss how to compose IPCs. In this aim, we introduce the language IPC_L, expressive enough to depict any IPC. We use $p, p_1, \ldots$ to range over probabilities (i.e., over $]0,1]$), $a, a_1, \ldots$ to denote actions in $\mathcal{A}$, and $B, B_1, \ldots$ for behaviors denoting an expression of the language.

Definition 5.2 (Language IPC_L)**.** *Behaviors of the language* IPC_L *are described by the grammar:*

$$B \ ::= \ \delta \mid a\,;B \mid \textstyle\sum_{\bullet i} p_i :: B_i \mid B_1 \,[\,]\, B_2 \mid B_1 \,|[A]|\, B_2 \mid \mathsf{hide}\ A\ \mathsf{in}\ B \mid \widetilde{B}$$

where A *is a set of actions not including* τ*,* $A \subset \mathcal{A} \setminus \{\tau\}$*.*

We write $\mathcal{B}$ to denote the set of all possible behaviors B. The operators of the language will be referred to as:
– The termination symbol δ denotes a *blocked behavior*.
– *Action-prefix* "$a\,;B$" denotes the *sequential composition* of the action a and the behavior B.
– The expression "$\sum_{\bullet i} p_i :: B_i$" denotes a *probabilistic choice*. It verifies the constraint $\sum_i p_i = 1$, ensuring that the probabilistic choice is well-defined.
– The expression "$B_1 \,[\,]\, B_2$" denotes a *nondeterministic choice* between B_1 and B_2.
– The expression "$B_1 \,|[A]|\, B_2$" denotes the *parallel composition* with synchronization set A (in LOTOS-style [BB87]).
– The expression "hide A in B" denotes abstraction and is an *hidden behavior*.
– Finally, $\widetilde{B}$ denotes a call to the process $\widetilde{B}$.

Processes $\widetilde{B}$ can be defined as $\widetilde{B} = B$, requiring recursion to be guarded, as in [BFP01]. The semantics of the language IPC_L is defined in a structured operational semantics style [JAB01] by mapping each behavior B onto an IPC.

Definition 5.3 (Semantics of IPC_L behaviors)**.** *The operational semantics of a behavior* B_0 *over the set of actions* $\mathcal{A}$ *is defined as the IPC* $P = \langle \mathcal{B}, \mathcal{A}, \longrightarrow, \rightsquigarrow, B_0 \rangle$*, where* $\longrightarrow$ *and* $\rightsquigarrow$ *are defined by the inference rules of figure 5.1.*

Because each term of IPC_L can be mapped on an IPC, we define the same operators as operators of IPC_L, with the same semantics for IPCs. For instance, "$P_1 \,|[A]|\, P_2$" denotes the parallel synchronization of two IPCs P_1 and P_2 with synchronization on actions of the set $A \in \mathcal{A}$.

		$\dfrac{}{\delta \overset{1}{\rightsquigarrow} \delta}$	(1)
		$\dfrac{}{\sum_{\bullet i} p_i :: B_i \overset{p_i}{\rightsquigarrow} B_i}$	(2)
$\dfrac{}{a;B \xrightarrow{a} B}$	(3.a)	$\dfrac{}{a;B \overset{1}{\rightsquigarrow} a;B}$	(3.b)
$\dfrac{B_1 \xrightarrow{a} B_1'}{B_1[\,]B_2 \xrightarrow{a} B_1'} \quad \dfrac{B_2 \xrightarrow{a} B_2'}{B_1[\,]B_2 \xrightarrow{a} B_2'}$	(4.a)	$\dfrac{B_1 \overset{p_1}{\rightsquigarrow} B_1' \quad B_2 \overset{p_2}{\rightsquigarrow} B_2'}{B_1[\,]B_2 \overset{p_1\,p_2}{\rightsquigarrow} B_1'[\,]B_2'}$	(4.b)
$\dfrac{B_1 \xrightarrow{a} B_1' \quad a \notin A}{B_1 \lvert [A] \rvert B_2 \xrightarrow{a} B_1' \lvert [A] \rvert B_2} \quad \dfrac{B_2 \xrightarrow{a} B_2' \quad a \notin A}{B_1 \lvert [A] \rvert B_2 \xrightarrow{a} B_1 \lvert [A] \rvert B_2'} \quad \dfrac{B_1 \xrightarrow{a} B_1' \quad B_2 \xrightarrow{a} B_2' \quad a \in A}{B_1 \lvert [A] \rvert B_2 \xrightarrow{a} B_1' \lvert [A] \rvert B_2'}$	(5.a)	$\dfrac{B_1 \overset{p_1}{\rightsquigarrow} B_1' \quad B_2 \overset{p_2}{\rightsquigarrow} B_2'}{B_1 \lvert [A] \rvert B_2 \overset{p_1\,p_2}{\rightsquigarrow} B_1' \lvert [A] \rvert B_2'}$	(5.b)
$\dfrac{B_1 \xrightarrow{a} B_1' \quad a \notin A}{\mathsf{hide}\ A\ \mathsf{in}\ B_1 \xrightarrow{a} \mathsf{hide}\ A\ \mathsf{in}\ B_1'} \quad \dfrac{B_1 \xrightarrow{a} B_1' \quad a \in A}{\mathsf{hide}\ A\ \mathsf{in}\ B_1 \xrightarrow{\tau} \mathsf{hide}\ A\ \mathsf{in}\ B_1'}$	(6.a)	$\dfrac{B_1 \overset{p}{\rightsquigarrow} B_1'}{\mathsf{hide}\ A\ \mathsf{in}\ B_1 \overset{p}{\rightsquigarrow} \mathsf{hide}\ A\ \mathsf{in}\ B_1'}$	(6.b)
$\dfrac{\widetilde{B}{=}B \quad B \xrightarrow{a} B'}{\widetilde{B} \xrightarrow{a} B'}$	(7.a)	$\dfrac{\widetilde{B}{=}B \quad B \overset{p}{\rightsquigarrow} B'}{\widetilde{B} \overset{p}{\rightsquigarrow} B'}$	(7.b)

Figure 5.1: Operational semantics of $\mathsf{IPC}_{\mathsf{L}}$

The rules presented in figure 5.1 are separated in two classes. The rules on the left (interactive rules) are inspired by LOTOS semantics and involve interactive transitions. The rules on the right (probabilistic rules) handle the probabilistic time extension. The intuitive meaning of this IPC_L semantics is the following:

- The blocked behavior δ cannot perform any action (including the internal τ action) and cannot interact anymore with its environment. There are consequently no interactive rules on the left for δ. However it does not prevent time from progressing (rule (1)).
- The action-prefix "$a\,;B$" may perform the action a and then behaves like B (rule (3.a)), or it can let time progress (rule (3.b)).
- The probabilistic choice behavior "$\sum_i^{\bullet} p_i :: B_i$" behaves, after one time step, like one of the B_i ; the chosen B_i is selected according to the probabilistic distribution over the B_i (rule (2)).
- The nondeterministic choice "$B_1 \,[\,]\, B_2$" depends on behaviors B_1 and B_2 (either they can take a probabilistic transition or they can take an interactive transition). We can distinguish three possibilities:
  - Either behavior B_1 can take an interactive transition, $B_1 \xrightarrow{a} B_1'$. In this case, "$B_1 \,[\,] B_2$" can take a transition $\xrightarrow{a}$ and behaves like B_1' (rules (4.a)).
  - Or behavior B_2 can take an interactive transition, $B_2 \xrightarrow{a'} B_2'$. In this case, "$B_1 \,[\,]\, B_2$" can take a transition $\xrightarrow{a'}$ and behaves like B_2' (rules (4.a)).
  - Or behaviors B_1 and B_2 can both take a probabilistic transition, $B_1 \overset{p_1}{\rightsquigarrow} B_1'$ and $B_2 \overset{p_2}{\rightsquigarrow} B_2'$. In this case, the nondeterministic choice is not resolved by time progression, i.e., "$B_1 \,[\,]\, B_2$" can behave, after one time step, like "$B_1' \,[\,]\, B_2'$" with a probability $p_1\ p_2$ (rules (4.b)).
- The behavior "$B_1 \,|[A]|\, B_2$" depends also on behaviors B_1 and B_2, and we can distinguish four possibilities:
  - Either behavior B_1 can take an interactive transition, $B_1 \xrightarrow{a} B_1'$, with $a \notin A$. In this case, "$B_1 \,|[A]|\, B_2$" can take a transition $\xrightarrow{a}$ and behaves like "$B_1' \,|[A]|\, B_2$" (rules (5.a)).
  - Or behavior B_2 can take an interactive transition, $B_2 \xrightarrow{a'} B_2'$, with $a' \notin A$. In this case, "$B_1 \,|[A]|\, B_2$" can take a transition $\xrightarrow{a'}$ and behaves like "$B_1 \,|[A]|\, B_2'$" (rules (5.a)).
  - Or behaviors B_1 and B_2 can both take an interactive transition, $B_1 \xrightarrow{a} B_1'$ and $B_2 \xrightarrow{a'} B_2'$, with $a = a'$ and $a \in A$. In this case, "$B_1 \,|[A]|\, B_2$" can take a transition $\xrightarrow{a}$ and behaves like "$B_1' \,|[A]|\, B_2'$" (rules (5.a)).
  - Or B_1 and B_2 can both take a probabilistic transition, $B_1 \overset{p_1}{\rightsquigarrow} B_1'$ and $B_2 \overset{p_2}{\rightsquigarrow} B_2'$. In this case, "$B_1 |[A]| B_2$" can behave, after one time step, like "$B_1' |[A]| B_2'$" with a probability $p_1 p_2$ (rule (5.b)).
- The hidden behavior "hide A in B" behaves like B for which all actions of A are replaced by τ (rules (6.a) and (6.b)).
- Finally the behavior $\widetilde{B} = B$ behaves like B (rules (7.a) and (7.b)).

Concerning the probabilistic rules, it is important to notice that time progresses synchronously. Indeed rules concerning nondeterministic choice (4.b) and parallel composition (5.b) of behaviors allowing to fire probabilistic transitions have their time progressing synchronously (for these binary operators, if a behavior takes a probabilistic transition, which takes one time step, the other behavior has to take synchronously a probabilistic transition).

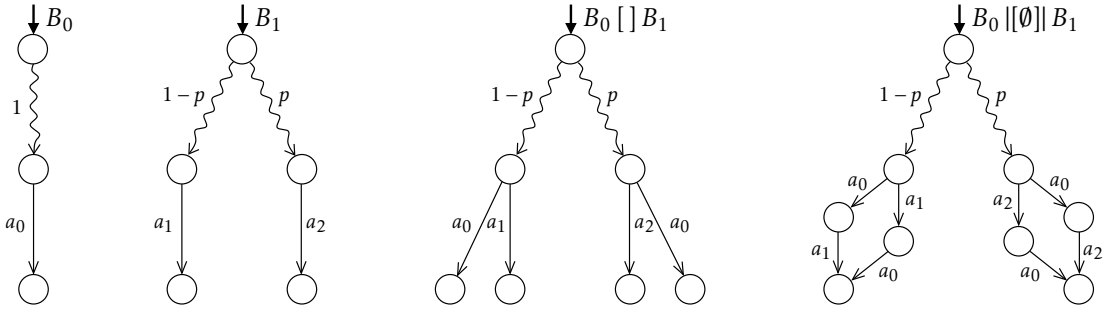


Figure 5.2: Synchronous time evolution for nondeterminism and parallel composition of probabilistic behaviors

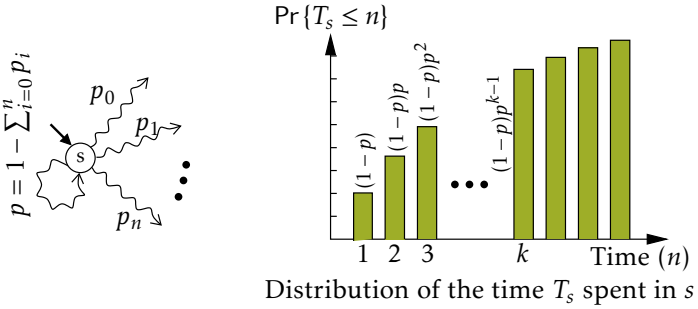


Figure 5.3: The time spent in each state of an IPC follows a geometric distribution

Example 5.1. *Consider the behaviors "*$B_0 = \sum\!\!\bullet 1 :: a_0 \,;\, \delta$*" and "*$B_1 = \sum\!\!\bullet \begin{pmatrix} 1-p :: a_1 \,;\, \delta \\ p :: a_2 \,;\, \delta \end{pmatrix}$*". When composing* B_0 *and* B_1 *(nondeterministic choice or parallel composition), time progresses synchronously as illustrated in figure 5.2. We can see that for* B_0 *and* B_1*, a time step has to elapse first. When composing them, we have this time step first, corresponding to the time progressing synchronously in the two processes: it is not possible for one of the processes to evolve alone in time.*

Despite time progressing synchronously, the sojourn time in each state of an IPC (i.e., the time spent in each state) follows a geometric distribution that present the memoryless property, as depicted in figure 5.3. If there is no probabilistic loop on a state s, the time spent in s is always equal to one, which is a limit behavior of the geometric distribution depicted in figure 5.3 with $p = 0$.

The distribution of the sojourn time in a state following a geometric distribution, each state present the memoryless property: whatever the time already spent in the state, the remaining sojourn time always follows the same geometric distribution.

Another important property is expressed by rules (1) and (3.b): a process can let time progress at any time. In particular, this property is important if the process is blocked (either because it is terminated or because it waits for synchronization that are currently not enabled). Indeed, although it is functionally blocked, it does not prevent other processes to advance in time. This property, called the *arbitrary waiting* is inspired by SOS rules of Hansson calculus [Han91, Han94]. In figures representing IPCs, we do not depict probabilistic transitions expressing the arbitrary waiting (i.e., loops with a probability one on states allowing to take only interactive transitions), but rather consider that those loops are implicit, saying that interactive

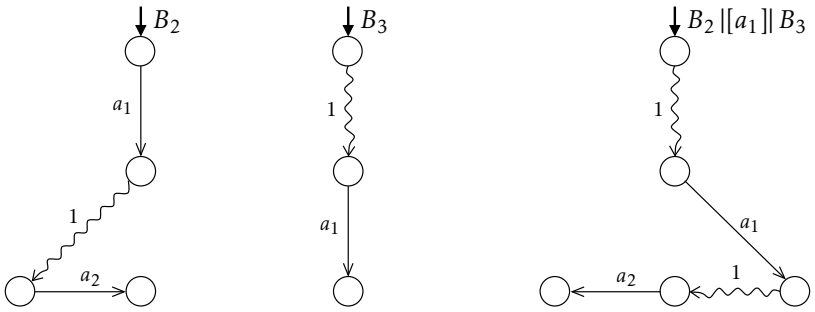


Figure 5.4: Illustration of the arbitrary waiting properties

transitions may be delayed.

Example 5.2. *Consider the behaviors "$B_2 = a_1 \,;\, \underline{\sum}\bullet 1 :: a_2 \,;\, \delta$" and "$B_3 = \underline{\sum}\bullet 1 :: a_1 \,;\, \delta$". The parallel composition of B_2 and B_3 with synchronization on a_1 ("$B_2|[a_1]|B_3$") is depicted in figure 5.4. Without the arbitrary waiting property (rule (3.b)), "$B_2 \,|[a_1]|\, B_3$" would be equivalent to δ: B_2 would be immediately blocked waiting for synchronization on a_1, preventing time from progressing in B_3. In the same way, after synchronization on a_1, without the arbitrary waiting property (rule (1)), "$B_2 \,|[a_1]|\, B_3$" would be equivalent to B_3, - hindering B_2 to go on evolving in time, because B_3 is blocked (δ).*

Finally notice that rule (4.b) expresses that, when time progresses, a nondeterministic choice between two states is not resolved but is postponed and resolved only when interactive transitions are possible later in time (according to rule (4.a)).

Example 5.3. *Consider the behaviors $B_4 = \underline{\sum}\bullet\begin{pmatrix} p :: B_4 \\ 1-p :: a \,;\, \delta \end{pmatrix}$ and $B_5 = \underline{\sum}\bullet\begin{pmatrix} q :: B_5 \\ 1-q :: b \,;\, \delta \end{pmatrix}$ depicted in figure 5.5. The sojourn time in initial states of B_4 and B_5 is geometrically distributed. Thus, whatever the time already spent in those initial states, the nondeterministic choice is not resolved. But also whatever the geometric distribution elapsing first, the nondeterministic choice "$B_4 \,[\,]\, B_5$" is not resolved. Indeed:*

- *either B_4 leaves its initial state first: the nondeterministic choice is not resolved immediately but only if the action a is fired by B_4 before B_5 leaves its initial state. Because of the arbitrary waiting property, a may be delayed, and, during this time slot, B_5 may leave its initial state. In this case, we have a nondeterministic choice between actions a and b.*
- *or, symmetrically, B_5 leaves its initial state first: the nondeterministic choice is not resolved immediately but only if the action b is fired by B_5 before B_4 leaves its initial state. Because of the arbitrary waiting property, b may be delayed, and, during this time slot, B_4 may leave its initial state. In this case, we have a nondeterministic choice between actions a and b.*
- *or B_4 and B_5 leave their initial states simultaneously, and the nondeterministic choice is also not resolved and remains between the reached states.*

Notice that for IMCs, we have a different behavior for nondeterministic choice: it is resolved on the minimum of the exponentially distributed sojourn times of the states between which there is a nondeterministic choice. The first change of state resolves the nondeterministic choice.

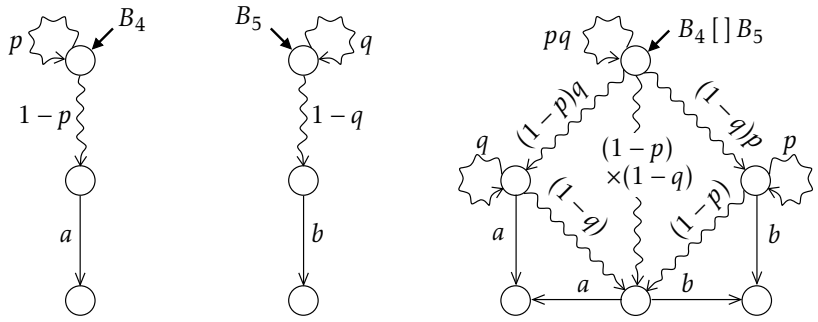


Figure 5.5: Nondeterministic choice between two probabilistic choices

5.1.2 Properties of Interactive Probabilistic Chains

We introduce several properties on IPCs. The first property deals with the competition between a τ-transition and a probabilistic choice in a state of an IPC.

Definition 5.4 (Maximal progress)**.** *An IPC $P = \langle S, \mathcal{A}, \longrightarrow, \rightsquigarrow, s_0 \rangle$ is said to be* maximal progress cut *if and only if*

$$\left(\forall (s, s') \in S^2\right) \quad s \xrightarrow{\tau} s' \Longrightarrow \left(\left(\nexists s'' \in S\right) \quad s \rightsquigarrow s''\right)$$

Given an IPC P, we call "maximal progress cut IPC of P", written as $P_{\not\rightarrow_\tau}$, the largest maximal progress cut IPC contained in P. The maximal progress property is sometimes also called minimal delay [Han91, Han94].

Example 5.4. *Consider the IPC P depicted in figure 5.6(a). P is not maximal progress cut: in state s_6, the τ-transition $s_6 \xrightarrow{\tau} s_7$ is in competition with the probabilistic transition $s_6 \overset{1}{\rightsquigarrow} s_4$. $P' = P_{\not\rightarrow_\tau}$, the maximal progress cut IPC of P is depicted in figure 5.6(b).*

The second property tackles the competition between arbitrary interactive transitions (including τ), and a probabilistic choice in an IPC.

Definition 5.5 (Urgency)**.** *An IPC $P = \langle S, \mathcal{A}, \longrightarrow, \rightsquigarrow, s_0 \rangle$ is said to be* urgency cut *if and only if*

$$\left(\forall (s, s') \in S^2\right)\left(\forall a \in \mathcal{A}\right) \quad s \xrightarrow{a} s' \Longrightarrow \left(\left(\nexists s'' \in S\right) \quad s \overset{p}{\rightsquigarrow} s''\right)$$

We can notice that an urgency cut IPC is by definition maximal progress cut. Given an IPC P, we call "urgency cut IPC of P", written as $P_{\not\rightarrow_{\mathcal{A}}}$, the largest urgency cut IPC contained in P.

Example 5.5. *Consider the IPC P depicted in figure 5.6(a). P is not urgency cut: in state s_8, the interactive transition $s_8 \xrightarrow{a_1} s_3$ is in competition with the probabilistic transition $s_8 \overset{1}{\rightsquigarrow} s_2$. $P'' = P_{\not\rightarrow_{\mathcal{A}}}$, the urgency cut IPC of P is depicted in figure 5.6(c).*

Finally, we introduce a property linked to the number of interactive transitions that can be taken in a finite time interval.

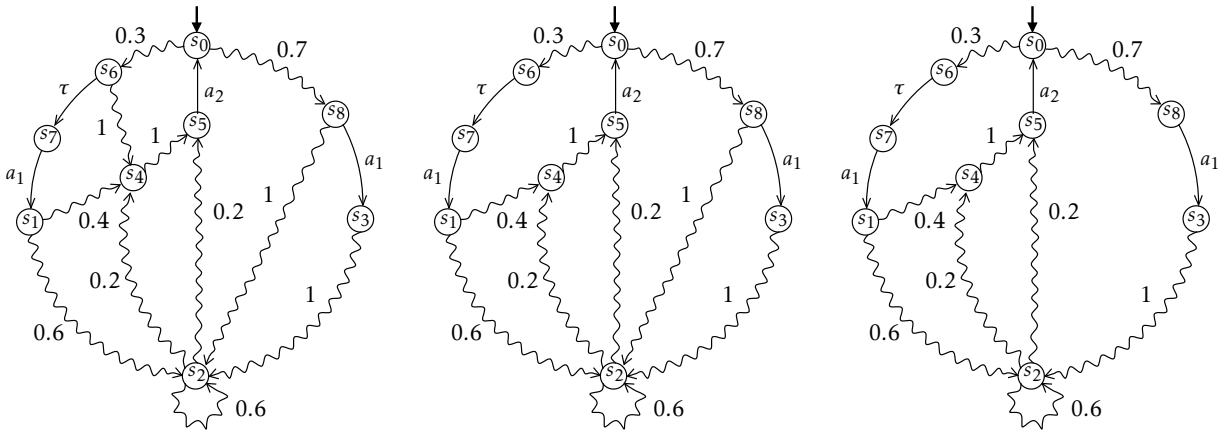


Figure 5.6: Illustration of maximal progress and urgency cut properties

Definition 5.6 (Non-Zeno)**.** *An IPC $P = \langle S, \mathcal{A}, \longrightarrow, \rightsquigarrow, s_0 \rangle$ is said to be* non-Zeno *if and only if it can perform only finitely many actions in a finite time interval. We ensure this property on IPCs by bounding the number of possible actions taken in sequence (i.e., in a time interval of one time step):*

$$\left(\forall (s,s') \in S^2\right) \left(\exists k \in \mathbb{N}\right) \left(\forall n \in \mathbb{N}\right) \left(\forall (s_1,\ldots,s_n) \in S^n\right)$$
$$s \longrightarrow s_1 \longrightarrow \cdots \longrightarrow s_n \longrightarrow s' \Longrightarrow (n \leq k)$$

The non-Zenoness property expresses that in a finite time interval, it is not possible to take an infinite number of interactive transitions. In particular, an IPC with loops of interactive transitions is not non-Zeno. The non-Zeno property allows to find a value k corresponding to the length of the longest sequence of interactive transitions in an IPC.

The definition of non-Zenoness property we gave is sometimes called *implementability* property [NS91]: the existence of a bound on the number of actions is ensured for any chosen time interval.

Example 5.6. *The IPC P depicted in figure 5.6(a) is non-Zeno. The length k of the longest sequence of interactive transition $s_6 \xrightarrow{\tau} s_7 \xrightarrow{a_1} s_1$ is $k = 2$.*

5.1.3 Probabilistic Strong and Branching Bisimulations

As seen for Markov chains in chapter 2, it is essential to have methods allowing to compare different processes. Comparing processes requires the definition of equivalence relations between processes. The bisimulations for labeled transition systems [vGW96, Mil90] or for DTMCs (a generalization for DTAMC was introduced in section 2.2.3) define equivalence relations allowing to compare different LTS or different DTMCs. A bisimulation for IPCs should cover the characteristics of both DTMCs and LTS, taking into account both probabilistic transitions of DTMCs and interactive transitions of LTS. We introduce bisimulations for IPCs along the lines of [HL98, LS92].

To define bisimulations, we extend the predicate γ_P introduced for probabilistic transitions of DTAMCs in section 2.2.3. Consider an IPC $P = \langle S, \mathcal{A}, \longrightarrow, \rightsquigarrow, s_0 \rangle \in \mathscr{P}$. The predicate $\gamma_P : S \times 2^S \mapsto [0,1]$ computes the cumulative probability to reach a set of states $\mathscr{S}$, from a state s:

$$\Big(\forall s \in S\Big)\Big(\forall \mathscr{S} \subseteq S\Big) \quad \gamma_P(s, \mathscr{S}) = \sum_{s' \in \mathscr{S}} \Big\{\!\!\Big[p \mid s \overset{p}{\rightsquigarrow} s' \Big]\!\!\Big\}$$

In addition, we introduce a new predicate γ_0 for interactive transitions. The predicate $\gamma_0 : S \times \mathcal{A} \times 2^S \mapsto \{\mathsf{True}, \mathsf{False}\}$ states if, from a given state, it is possible to reach a set of states with a given action, i.e.,

$$\Big(\forall s \in S\Big)\Big(\forall a \in \mathcal{A}\Big)\Big(\forall \mathscr{S} \subseteq S\Big) \quad \gamma_0(s, a, \mathscr{S}) = \begin{cases} \mathsf{True} & \text{if } \Big(\exists s' \in \mathscr{S}\Big) \quad s \xrightarrow{a} s' \\ \mathsf{False} & \text{otherwise} \end{cases}$$

Finally, we introduce some notations. $s \overset{\tau}{\not\longrightarrow}$ is an abbreviation for $\neg\gamma_0(s, \tau, S)$ and means that s is not able to fire a τ-transition. $\mathscr{P}/_{\mathscr{E}}$ denote the set of equivalence classes of $\mathscr{P}$ with respect to relation $\mathscr{E}$ and $[s]_{/_{\mathscr{E}}}$ denote the equivalence class with respect to $\mathscr{E}$ containing the state s.

Using those notations, one can define a strong probabilistic bisimulation that allows us to compare IPCs. Intuitively, the strong probabilistic bisimulation states that if two processes are strongly bisimilar, they have to functionally simulate each other and that the overall transition probabilities of each process to the same equivalence class are equal.

Definition 5.7 (Probabilistic strong bisimulation of IPC)**.** *Strong probabilistic bisimulation equivalence ($\sim$) is the coarsest equivalence relation on $\mathscr{P}$ such that $s_1 \sim s_2$ implies for all actions $a \in \mathcal{A}$ and all equivalence classes $\mathcal{C}$ of $\sim$ (i.e., $\mathcal{C} \in \mathscr{P}/_{\sim}$):*

(1) $\gamma_0(s_1, a, \mathcal{C}) \implies \gamma_0(s_2, a, \mathcal{C})$,

(2) $s_1 \overset{\tau}{\not\longrightarrow} \implies \Big(s_2 \overset{\tau}{\not\longrightarrow} \wedge\ \gamma_P(s_1, \mathcal{C}) = \gamma_P(s_2, \mathcal{C})\Big)$.

When comparing IPCs, one may want to abstract from internal computation, and thus internal transitions τ. We use a weaker notion of equivalence, branching bisimulation, here lifted to IPC [HL98, vGW96].

Definition 5.8 (probabilistic branching bisimulation of IPC)**.** *Probabilistic branching bisimulation equivalence ($\approx$) is the coarsest equivalence relation on $\mathscr{P}$ such that $s_1 \approx s_2$ implies for all action $a \in \mathcal{A}$ and all equivalence classes $\mathcal{C}$ of $\approx$ (i.e., $\mathcal{C} \in \mathscr{P}/_{\approx}$):*

(1) $\gamma_0(s_1, a, \mathcal{C}) \implies$ *either* $(a = \tau \wedge s_2 \in \mathcal{C})$ *or* $\Big(\exists s_2'\Big)\Big(s_2 \overset{\tau*}{\longrightarrow} s_2' \wedge s_1 \approx s_2' \wedge \gamma_0(s_2', a, \mathcal{C})\Big)$

(2) $s_1 \overset{\tau}{\not\longrightarrow} \implies \Big(\exists s_2' \overset{\tau}{\not\longrightarrow}\Big)\Big(s_2 \overset{\tau*}{\longrightarrow} s_2' \wedge s_1 \approx s_2' \wedge \gamma_P(s_1, \mathcal{C}) = \gamma_P(s_2', \mathcal{C})\Big)$

Two IPCs P_1 and P_2 are said to be strongly (resp. branching) bisimilar, if their initial states are strongly (resp. branching) bisimilar.

Lemma 5.1. *If two IPCs are strongly bisimilar, they are also branching bisimilar:*

$$\Big(\forall (P_1, P_2) \in \mathscr{P}^2\Big) \quad P_1 \sim P_2 \Longrightarrow P_1 \approx P_2$$

Proof. Consider two states s_1 and s_2 strongly bisimilar, $s_1 \sim s_2$, and an equivalence class $\mathcal{C} \in \mathscr{P}/_\sim$. By definition, if $\gamma_0(s_1,a,\mathcal{C})$ holds, then $\gamma_0(s_2,a,\mathcal{C})$ holds. Taking $s_2' = s_2$, condition *(1)* of the probabilistic branching bisimulation is verified.

On the same idea, if $s_1 \stackrel{\tau}{\not\longrightarrow}$, then $s_2 \stackrel{\tau}{\not\longrightarrow}$ and $\gamma_P(s_2,\mathcal{C}) = \gamma_P(s_1,\mathcal{C})$. Taking $s_2' = s_2$, condition *(2)* of the probabilistic branching bisimulation is verified. □

In addition to comparing IPCs, a bisimulation can be used to define the smallest IPC (in terms of state space size) equivalent to another one. This smallest equivalent IPC is called *quotient*. One may define a quotient according to the probabilistic strong bisimulation $\sim$ and a quotient according to the probabilistic branching bisimulation $\approx$.

Definition 5.9 (IPC quotient)**.** *Given an IPC $P = \langle S, \mathcal{A}, \longrightarrow, \rightsquigarrow, s_0\rangle$ and a bisimulation $\mathscr{E}$, the quotient of P according to $\mathscr{E}$ is the IPC $P_{/_\mathscr{E}} = \langle S_{/_\mathscr{E}}, \mathcal{A}, \longrightarrow_{/_\mathscr{E}}, \rightsquigarrow_{/_\mathscr{E}}, [s_0]_{/_\mathscr{E}}\rangle$. $P_{/_\mathscr{E}}$ verifies:*

- *$P_{/_\mathscr{E}}$ and P are bisimilar ($P_{/_\mathscr{E}}$ $\mathscr{E}$ P)*
- *for every IPC P', defined over the state space S', and bisimilar to P (and to $P_{/_\mathscr{E}}$), P' $\mathscr{E}$ P, we have $\mathsf{card}(S_{/_\mathscr{E}}) \leq \mathsf{card}(S')$*

We say that an IPC is minimal (with respect to a bisimulation $\mathscr{E}$) if its size is the same as the size of its quotient. The quotient of an IPC can be computed similarly to the quotient of an IMC, using partition refinement techniques [Her02], as implemented in the BCG_MIN tool [INR].

In the previous section, we introduced maximal progress cut and urgency cut allowing to define subclasses of IPCs.The maximal progress cut presents an interesting property according to the bisimulations: maximal progress cut preserves the strong probabilistic bisimulation.

Lemma 5.2. *An IPC P and its maximal progress cut IPC $P_{\not\rightarrow_\tau}$ are strongly probabilistic bisimilar:*

$$\left(\forall P \in \mathscr{P}\right) \quad P \sim P_{\not\rightarrow_\tau}$$

Proof. The condition *(2)* of the strong probabilistic bisimulation is only defined for states that do not allow to take a τ-transition. Because a maximal progress cut IPC $P_{\not\rightarrow_\tau}$ only differs from P on states allowing to take a τ-transition, this property is ensured. □

Contrary to the maximal progress, the urgency cut does not preserve bisimulations. Indeed by cutting probabilistic transitions that compete with interactive ones, the condition *(2)* of the probabilistic strong and branching bisimulations are not ensured.

As corollary, lemma 5.2, together with lemma 5.1, implies that maximal progress cut preserves the probabilistic branching equivalence:

Lemma 5.3. *An IPC P and its maximal progress cut IPC $P_{\not\rightarrow_\tau}$ are branching equivalent:*

$$\left(\forall P \in \mathscr{P}\right) \quad P \approx P_{\not\rightarrow_\tau}$$

Proof. Direct implication from lemma 5.2 and lemma 5.1. □

Finally, we can deduce that the quotient of an IPC with respect to the probabilistic branching bisimulation is also maximal progress cut.

Lemma 5.4. *The quotient $P_{/\approx}$ of an IPC P is maximal progress cut:*

$$\left(\forall P \in \mathscr{P}\right) \quad \left(P_{/\approx}\right)_{\not\rightarrow_\tau}$$

Proof. By definition, for every IPC P, the state space of $P_{\not\rightarrow_\tau}$ is smaller or equal to the state space of P. In particular, given an IPC P, the state space of $\left(P_{/\approx}\right)_{\not\rightarrow_\tau}$ is smaller or equal to the state space of $P_{/\approx}$.

The definition of a quotient (definition 5.9) ensures that for every IPC P' branching bisimilar to $P_{/\approx}$, the state space of $P_{/\approx}$ is smaller or equal to the state space of P'. Because $\left(P_{/\approx}\right)_{\not\rightarrow_\tau} \approx P_{/\approx}$ (lemma 5.3) we have that $\left(P_{/\approx}\right)_{\not\rightarrow_\tau}$ and $P_{/\approx}$ have the same state space: the quotient is consequently maximal progress cut. □

5.1.4 Congruence Property of Probabilistic Branching Bisimulation

One may be interested in the preservation of the bisimulations with respect to the composition operators of IPC_L. Indeed, it is interesting to know if applying a same operator (sequence, composition, etc.) of IPC_L on two equivalent IPCs yields two equivalent IPCs. This so-called *congruence* property is of importance for the parallel composition of IPCs ($|[\cdots]|$). Indeed, the state space explosion may mainly occur when composing processes in parallel. In this case, it is interesting to reduce state space of processes before composing them.

Theorem 5.1 (Congruence with respect to $|[\cdots]|$)**.** *The probabilistic branching bisimulation is a congruence with respect to the parallel composition operator, i.e.,*

$$\left(\forall (P_1, P_2) \in \mathscr{P}^2\right)\left(\forall P_3 \in \mathscr{P}\right)\left(\forall A \subseteq \mathcal{A}\backslash\{\tau\}\right) \quad P_1 \approx P_2 \Longrightarrow P_1 \,|[A]|\, P_3 \approx P_2 \,|[A]|\, P_3$$

Proof. See appendix B. □

The congruence property with respect to composition operators is mandatory for a compositional approach, i.e., constructing large systems hierarchically. Indeed, in a compositional approach, a complex system is obtained by iteratively composing subcomponents (e.g., using the parallel composition operator for IPCs). The congruence property ensures that the composition of the quotients of subcomponents is bisimilar to the direct composition of subcomponents. In practice, it is preferable to compose the quotients: the state space of the composition of the quotients is at most equal to the state space of the direct composition of the subcomponents. Thus, composing quotients may be possible for a system where composing subcomponents leads to state space explosion. If a compositional approach allows to go farther and handle larger systems, we are not prevented from the state space explosion.

5.2 From Interactive Probabilistic Chains to Discrete-Time Annotated Markov Chains

In chapter 2, we introduced methodologies to extract performance measures from AMCs. The weakness of those methodologies concerns the way these annotations are obtained. Indeed, for complex Markov chains with a large state space, it is difficult, not to say impossible, to

associate functional information to states. On the other hand, IPC models allow to depict a complex system functionally and temporally combining characteristics of interactive processes and DTMCs.

In this section, we present a way to adapt DTMC-oriented performance methodologies to IPCs. In most cases, the underlying performance model of an IPC is a Markov decision process [Put94] (MDP), i.e., a DTMC with nondeterminism, and DTMC-oriented performance methods are inapplicable directly. To use those methods,some restrictions on IPCs are required, to ensure that their underlying model is a DTMC. The restrictions imposed on the IPC models we are using for performance evaluation lead to the class of time deterministic interactive probabilistic chains (tdIPCs), which are IPCs presenting determinism in time. tdIPCs can be used to compute performance measures for the modeled systems.

tdIPCs can be transformed into DTMCs, which permits the use of DTMC-oriented performance methodologies. This transformation is addressed by using notions of maximal progress and urgency cut defined in the previous section. More precisely, to keep functional information of IPCs, the proposed transformation leads to annotated Markov chains. This kind of transformation must preserve timed and probabilistic behaviors of the system. In other words, performance measures extracted from two bisimilar IPCs have to be the same. We show that this is the case for our transformation.

5.2.1 Alternating Interactive Probabilistic Chain

Up to now, we considered arbitrary IPCs. When looking at SOS rules introduced in section 5.1.1, we can notice that we do not give a meaning to competition between interactive transitions and probabilistic choices, although it is an encountered case for every states allowing to take an interactive transition. The SOS Rules are as permissive as possible, i.e., there is no a priori choice whether interactive or probabilistic transitions should be taken first. In a time perspective, it means that an interactive transition can be delayed as long as wanted. This property allows to cover all possible timed behaviors, but is annoying for performance evaluation: no measure can be computed because everything may happen.

Example 5.7. *Consider the Behavior B of* $\mathbb{IPC}_L$*, "*$B = a\,;\delta$*". The arbitrary waiting property ensures that the action "a" may be delayed for an arbitrary long time (for instance, "a" may be delayed until a synchronization on "a" is allowed). Consequently, possible performance results are not interesting. For instance, the time before the end of the process (*δ*) can take any value on* $\mathbb{N}$.

In the aim of performance evaluation, we are thus forced to give a meaning to competition between interactive transitions and probabilistic transitions. We start with competition between τ-transitions and a probabilistic choice: we assume that a process cannot delay an internal transition. This assumption is justified by the fact that a τ-transition can be immediately taken (because it cannot be delayed by the environment) and has no reason to be delayed by the process. Conversely, a probabilistic choice always incurs a one time step delay. Consequently, a τ-transition has precedence over any probabilistic transition. Considering an IPC P, the precedence of τ-transition over probabilistic transitions is ensured by taking the maximal progress (definition 5.4) cut IPC $P_{\not\rightarrow_\tau}$ of P. We also saw that the maximal progress cut is realized by the quotient of an IPC according to the probabilistic branching bisimulation.

The maximal progress defined for IPCs matches with the maximal progress used in the IMC formalism (the justification is rather similar: Markovian transitions incur a non-zero delay

versus immediate τ-transitions). Because maximal progress cut preserves branching equivalence, it can be applied at any intermediate phase of the compositional approach.

Following similar ideas, we handle the competition between interactive transitions (different from τ) and probabilistic transitions. Actually, we have no a-priori information concerning interaction between an isolated process and its environment. Consequently, an interactive transition may be delayed by the environment for an arbitrary long time. For instance, an interactive transition may be delayed, waiting for a synchronization to be enabled. If the synchronization is not enabled immediately, only the probabilistic choice is possible. Applying maximal progress for arbitrary interactive transitions (different from τ) would prevent this desirable behavior.

When analyzing a fully modeled system, we consider that it is *closed*, which means that there is no more possible interaction with its environment (including synchronizations). In other words, an interactive transition has no more reason to be delayed in time and is always enabled. Under this assumption of a closed system model, maximal progress can be generalized to all actions. It corresponds to the urgency cut (definition 5.5).

Application of the urgency cut on a closed IPC partitions the state space in two classes. A state is said to be *interactive* if it only allows interactive transitions to be fired. Conversely, a state is said to be *probabilistic* if it only allows a unique probabilistic choice between reachable states.

Definition 5.10 (Partitioning of the state space by urgency cut)**.** *Given an urgency cut IPC P, $P = \langle S, \mathcal{A}, \longrightarrow, \rightsquigarrow, s\rangle$, the state space is partitioned in to classes S_I and S_P, i.e., $S = S_I \cup S_P$ and $S_I \cap S_P = \emptyset$. The classes S_I and S_P are defined by:*

- *S_I is the class of interactive states, $\left(\forall s \in S_I\right)\left(\nexists s' \in S\right) \quad s \rightsquigarrow s'$.*
- *S_P is the class of probabilistic states, $\left(\forall s \in S_P\right)\left(\nexists s' \in S\right) \quad s \longrightarrow s'$.*

Example 5.8. *The IPC $P_{\not\rightarrow_{\mathcal{A}}}$ depicted in figure 5.6(c) is urgency-cut. Its state space is partitioned in two classes. The class of interactive states is $S_I = \{s_5, s_6, s_7, s_8\}$. The class of probabilistic states is $S_P = \{s_0, s_1, s_2, s_3, s_4\}$.*

This partitioning of the state space yields a so-called *alternating* model [And00, Han94, Han91, NS91], referring to the fact that during an execution, the current state is either interactive or probabilistic, but cannot be both interactive and probabilistic. The alternation between interactive and probabilistic states is not strict as imposed in some models [Han91].

The alternation between actions and probabilities leads to a system where evolutions in time and functional evolutions are clearly separated. Either state are interactive, and actions are taken, but time does not progress. Or states are probabilistic, and the next state is chosen according to the probabilistic distribution, this choice resulting in a one step progress in time. This view is adopted by most of the existing description formalisms for timed systems [NS91].

In addition, applying urgency cut to bisimilar processes yields bisimilar urgency-cut processes.

Lemma 5.5. *The two urgency cut IPCs of two branching bisimilar IPCs are branching bisimilar:*

$$\left(\forall (P_1, P_2) \in \mathscr{P}\right) \quad (P_1 \approx P_2) \Longrightarrow \left(P_{1\not\rightarrow_{\mathcal{A}}} \approx P_{2\not\rightarrow_{\mathcal{A}}}\right)$$

Proof. See appendix A. □

5.2.2 Dealing With Nondeterminism in Interactive Probabilistic Chains

An urgency-cut IPC remains in general nondeterministic. Nondeterministic choices are only possible in interactive states. In fact, nondeterminism is an essential aid in building interacting systems from components. R. Segala distinguishes three different uses of nondeterminism in system modeling [Seg95]:

- **External environment.** This kind of nondeterminism exploits the benefits of the compositional approach. It is used to simply define the behavior of components of the system. Indeed, for each component, all interactions with the environment (the other components) are defined without knowledge of the environment's behavior. In a closed system (i.e., all components are present and there is no more interaction with the environment), the component should be constrained by its environment resulting in a deterministic behavior.
- **Implementation freedom.** The nondeterminism is used to depict the specification of a component and not its implementation. In other words, nondeterminism allows to depict what the component has to do and not how it does it (e.g., modeling an arbitration policy by nondeterminism depicts that there is an arbitration but not exactly how this arbitration is implemented). This kind of nondeterminism implies that several choices are possible.
- **Scheduling freedom.** Nondeterminism is classically used to depict systems evolving in parallel. In IPC models, parallel functional behaviors are interleaved. After hiding actions occurring in parallel, the complete system however often shows deterministic behavior modulo branching bisimulation equivalence. In this case, transitions that are involved in the nondeterministic choice are confluent to a single common future. From a timed view of the system, whatever the chosen path to the common future, all actions are taken at the same time instant. This kind of nondeterminism can be solved by the multiset of fired actions: this multiset is the same for all possible paths to the common future.

In the following, we present how these kinds of nondeterminism should be handled.

Dealing With Nondeterminism Due to External Environment

The compositional approach exploits frequently nondeterminism due to external environment. As soon as the model of a component of the system wants to synchronize on a given transition, the transition is enabled. When composing on this transition with another component (the environment), it may not remain enabled.

For performance evaluation purpose we focus in the analysis of IPCs that do not present nondeterminism due to external environment. This is a partial solution targeting only closed systems (for instance, we are not able to compute performance measures on a subcomponent of the system whatever the behavior of its environment is). However computed performance measures are much more exploitable than for non-closed systems.

Example 5.9. *Consider the Behavior B of IPC_L, "$B = a;\delta$" studied in example 5.7. We saw that the arbitrary waiting property ensures that the action a may be delayed for an arbitrary long time. For instance, when studying the time before the end of the process (δ), the result is not really interesting: it can take any value on $\mathbb{N}$.*

Now, supposing that the system is closed, actions are ensured to have no reason to be delayed

(which allows to apply urgency cut). In this case, the time before the end of the process (δ) is exactly equal to 0*.*

In closed systems, no interaction is possible with the environment and the urgency cut can be applied. If non determinism remains on an urgency cut IPC, it is not due to external environment.

The use of urgency cut on closed systems induces a second restriction on the class of studied systems. We restrict our study to non-Zeno systems. Indeed, a Zeno IPC implies that there is a possible loop of interactive transitions. After urgency-cut, this loop represents a time-lock: when reaching the loop of interactive transitions, the process is live-locked in it, with time no longer progressing. The non-Zeno property is an essential requirement ensuring that the IPC depicts a realistic system.

Dealing With Nondeterminism Due to Implementation Freedom

The nondeterminism due to implementation freedom is very useful for functional modeling of large systems. Indeed, it is an efficient way of abstraction of complex behaviors (having a large state space). In the field of functional verification, a nondeterministic abstraction generally subsumes more behaviors than the real system allows, but may have no impact on computed functional results. For instance, a functional safety property, verified on a system allowing a large set of behaviors, is also verified on a system allowing a subset of behaviors of the first system.

In the field of performance evaluation, the way to solve nondeterminism due to implementation freedom may influence the performance measures. We consequently say that this kind of nondeterminism is due to under-specifications (although they may have been deliberately introduced in the functional models).

In the sequel, we limit our performance study to systems without nondeterminism due to implementation freedom. When encountering this kind of nondeterminism, the system has to be determinized, i.e., fully specified.

Unfortunately, the determinization of a system using standard algorithm of automata theory [RS59] may increase its state space. Another approach that does not increase the state space is to replace each nondeterministic choice by a probabilistic choice. However, this approach would yield models that cannot be easily represented as IPCs. Indeed we would have to distinguish two kinds of probabilistic choices: the ones incurring zero time elapsing (thus corresponding to the probabilistically determinized nondeterministic choices) and the ones incurring one time step. Furthermore, this approach influences the performance measures as illustrated by the following example.

Example 5.10. *Consider a producer feeding 2 consumers through an arbiter. A simple functional model of the arbiter would be a nondeterministic choice between the 2 consumers. It would mean that an element produced would be either send to the first consumer, or to the second one.*

If we want to evaluate the performance of the system, the arbitration policy has to be modeled. For instance consider a round-robin policy. The functional model of this policy is larger than a single nondeterministic choice, because we have to store which consumer was fed last. At the cost of a larger model, the implementation freedom nondeterminism is deleted.

Because the round-robin policy is fair, another solution would be to turn the nondeterministic

choice into an equiprobable probabilistic choice. This solution allows to keep the same state space as with nondeterminism, but in a performance context.

*However, as for nondeterminism in the functional model, this kind of probabilistic choice is an abstraction and may impact performance results. Consider one of the two consumers: The probability to receive two consecutive elements from the arbiter is non-null (*0.25*) although it is null for the real behavior with modeled round-robin.*

Thus, we do not consider the approach consisting in replacing nondeterminism by probabilistic choices in the rest of this chapter.

Dealing With Nondeterminism Due to Scheduling freedom

Nondeterminism due to scheduling freedom arises from the interleaving of parallel functional behaviors. In an IPC, we have asynchronous parallelism of actions that take no time between phases of synchronous evolution in time. As said previously, for this kind of nondeterminism, the resulting system is deterministic modulo probabilistic branching bisimulation equivalence after hiding all actions.

The determinism modulo branching equivalence is also exploited to generate CTMCs from IMCs: abstracting from all functional information (i.e., renaming interactive transitions into τ), and minimizing with respect to IMC branching bisimulation equivalence, yields CTMCs (in the case the nondeterminism is only due to scheduling freedom).

We could imagine to exploit the same property on IPCs abstracting from all functional information and minimizing with respect to IPC branching bisimulation equivalence. Unfortunately, this method *always* yields the very same deterministic system: a single state with a probabilistic looping transition with probability 1. This single state IPC is obtained even if nondeterminism is not only due to scheduling freedom. However, this result is predictable: from a timed perspective, the only observable thing in an IPC having all interactive transitions hidden, is the time evolving discretely. This means that IPC branching bisimulation equivalence can only be used to reduce the IPC state space by abstracting from meaningless functional information. This minimization may delete some scheduling freedom-related nondeterminism, but in general, an IPC keeping significant functional transitions may stay non-deterministic. We consequently allow IPCs to present nondeterminism due to scheduling freedom in the sequel.

5.2.3 Time Deterministic and Deterministic Interactive Probabilistic Chain

To evaluate performance of a system, we introduced several restrictions on its IPC model. First, the considered IPC has to be non-Zeno and closed, to allow the application of the urgency cut. Then, the studied IPC has to be free of nondeterminism due to implementation freedom (it has to be fully specified). With those restrictions, the only determinism allowed is the one linked to scheduling freedom. However, the IPC remains time deterministic, nondeterminism not affecting the evolution in time.

We call this kind of IPCs *time deterministic* IPCs (tdIPCs). To provide a formal definition of tdIPCs, we first introduce some notations. Consider $\mathfrak{E}$ the set of regular expressions over a set of actions $\mathcal{A}$. we define $\mathfrak{e}$ as the empty expression, denoting that there is no action. for a regular expression $e \in \mathfrak{E}$, let $s_0 \xrightarrow{e} s_n$ denote that there exists a sequence of interactive

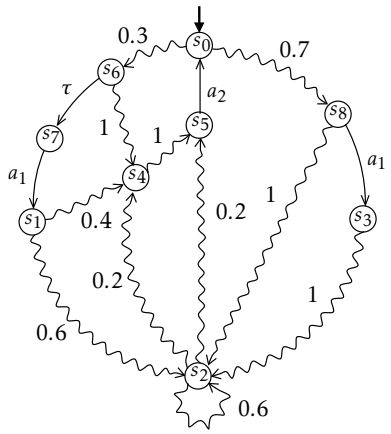


Figure 5.7: a (time) deterministic IPC of a closed system

transitions $s_0 \xrightarrow{a_1} \dots \xrightarrow{a_n} s_n$ where $a_1 \dots a_n$ is a word in the language over $\mathcal{A}$ defined by e. We will note $\mathfrak{S}(e)$ the multiset of actions $\{a_i \mid a_i \neq \tau\}_{1 \leq i \leq n}$ induced by the word $a_1 \dots a_n$ defined by e. the empty expression $\mathfrak{e}$ verifies that $s_1 \xrightarrow{\mathfrak{e}} s_2$ implies that $s_1 = s_2$ and $\mathfrak{S}(\mathfrak{e}) = \emptyset$. A tdIPC is characterized by a property on the largest urgency cut IPC contained therein.

Definition 5.11 (Time deterministic interactive probabilistic chain)**.** *An IPC is said to be a time deterministic IPC (tdIPC), if it is closed, non-Zeno, and the largest urgency-cut IPC contained therein, $P = \langle S = S_I \cup S_P, \mathcal{A}, \longrightarrow, \rightsquigarrow, s \rangle$, verifies:*

$$\left(\forall s_1 \in S_I\right)\left(\forall s_2 \in S_P\right)\left(\forall (e, e') \in \mathfrak{E}^2\right) \quad \left(s_1 \xrightarrow{e} s_2 \quad \wedge \quad s_1 \xrightarrow{e'} s_2\right) \Longrightarrow (\mathfrak{S}(e) = \mathfrak{S}(e'))$$

A tdIPC is a model including functional informations and time informations. Functional nondeterminism has no influence on its evolution in time and thus on timed characteristics that can be used to compute performance measures. In [CHLS09], we considered a stronger restriction than time determinism to compute performance measures. Indeed, we imposed that IPCs would have to be *deterministic*.

Definition 5.12 (Deterministic interactive probabilistic chain)**.** *An IPC $P = \langle S = S_I \cup S_P, \mathcal{A}, \longrightarrow, \rightsquigarrow, s_0 \rangle$ is called a deterministic IPC (dIPC), if it is urgency-cut and it verifies:*

$$\left(\forall (s, s', s'') \in S^3\right)\left(\forall (a, a') \in \mathcal{A}^2\right) \quad \left(s \xrightarrow{a} s' \wedge s \xrightarrow{a'} s''\right) \Longrightarrow (s' = s'' \wedge a = a')$$

A deterministic IPC guarantees that only linear sequences of interactive transitions may appear. Compared to a time deterministic IPC, there is no nondeterminism due to implementation freedom. That is why a dIPC is also a tdIPC.

Example 5.11. *The IPC P depicted in figure 5.6(a) and redrawn in figure 5.7 is non-Zeno. If we assume that it is closed, it is also a dIPC (and thus a tdIPC) because there is no nondeterminism.*

Note that a time deterministic IPC can be scheduled arbitrarily to be transformed into a deterministic IPC. By this way, we give the priority to one of the possible scheduling of parallel functional behaviors.

Definition 5.13 (Scheduler on a time deterministic interactive probabilistic chain)**.** *Given an IPC* $P = \langle S = S_I \cup S_P, \mathcal{A}, \longrightarrow, \rightsquigarrow, s_0 \rangle$*, a scheduler* $\mathfrak{s}$ *on* P *is a function that associates to each interactive state one of the interactive transition that can be taken, i.e.,*

$$\begin{array}{rccl} \mathfrak{s}: & S_I & \mapsto & \longrightarrow \\ & \mathfrak{s}(s) & = & \text{“}\, s \xrightarrow{a} s' \,\text{”} \end{array}$$

for some $a \in \mathcal{A}$ *and* $s \in S$*. A scheduler associates to each interactive state one of the interactive transition that can be taken.*

Lemma 5.6 (Scheduled time deterministic interactive probabilistic chain)**.** *Given an IPC* $P = \langle S = S_I \cup S_P, \mathcal{A}, \longrightarrow, \rightsquigarrow, s_0 \rangle$ *and a scheduler* $\mathfrak{s}$ *on* P*, we define the IPC* $\mathfrak{s}(P)$ *as the IPC obtained by scheduling of* P *by* $\mathfrak{s}$*.* $\mathfrak{s}(P)$ *is a deterministic IPC.*

Proof. For every state $s \in S_I$, the scheduler $\mathfrak{s}$ gives the priority to one of the interactive transition. There is consequently no more nondeterminism, which ensures that $\mathfrak{s}(P)$ is deterministic. □

We propose a translation of tdIPC (and thus dIPC) models into DTAMCs, preserving on one side the timed and probabilistic characteristics and on the other side functional information (actions of interactive transitions) by annotation of states.

5.2.4 Interactive Probabilistic Chain to Discrete Time Annotated Markov Chain

The transformation from a tdIPC P to DTAMC processes in two steps: first, P is turned into the largest urgency-cut IPC contained therein, $P_{\not\rightarrow_{\mathcal{A}}}$. Then, $P_{\not\rightarrow_{\mathcal{A}}}$ is transformed in a DTAMC, in such a way that functional information (the interactive transitions) appears in the annotation given to each state: intuitively, each probabilistic state of $P_{\not\rightarrow_{\mathcal{A}}}$ is enriched with functional information corresponding to the set of actions taken since the last time step, i.e., since the last probabilistic transition.

We define $\mathcal{P}^{\times}(\mathcal{A})$ as the set of sub-multisets of $\mathcal{A}$ and $\mathcal{P}^{\times}_{\leq k}(\mathcal{A})$ the set of sub-multisets of $\mathcal{A}$ containing at most k actions, i.e.,

$$\mathcal{P}^{\times}_{\leq k}(\mathcal{A}) = \{\mathscr{S}_{\times} \in \mathcal{P}^{\times}(\mathcal{A}) \mid \mathsf{card}(\mathscr{S}_{\times}) \leq k\}$$

Definition 5.14 (DTAMC associated to an IPC)**.** *Let* $P = \langle S = S_I \cup S_P, \mathcal{A}, \longrightarrow, \rightsquigarrow, s \rangle$ *be an urgency cut tdIPC over* $\mathcal{A}$*. Let* k *be the length of the longest sequence of interactive transitions in* P *(existence of* k *is ensured by definition 5.6 of a non-Zeno IPC). The DTAMC* $\mathcal{M}(P) = \langle C, \rightsquigarrow, c, \mathscr{A} \rangle$ *associated to* P *is such that:*

- *its set of states* $C = \left\{s \in R_P(P)\right\} \times \mathcal{P}^{\times}_{\leq k}(\mathcal{A})$*, where* $R_P(P) \subseteq S_P$*, is the set of reachable probabilistic states*
- $\rightsquigarrow \subset C \times]0,1] \times C \to \mathbb{N}$ *is a multiset of probabilistic transitions defined by:*

$$\begin{array}{l} \Big(\forall \mathscr{S}_1, \mathscr{S}_2 \in \mathcal{P}^{\times}_{\leq k}(\mathcal{A})\Big)\Big(\forall (s_1, s_2) \in R_P(P)^2\Big) \\ \langle s_1, \mathscr{S}_1 \rangle \overset{p}{\rightsquigarrow} \langle s_2, \mathscr{S}_2 \rangle \Longleftrightarrow \left(\begin{array}{l} \Big(\exists p \in]0,1]\Big)\Big(\exists s_1' \in S\Big)\Big(\exists e \in \mathfrak{E}\Big) \\ s_1 \overset{p}{\rightsquigarrow} s_1' \xrightarrow{e} s_2 \wedge \mathfrak{S}(e) = \mathscr{S}_2 \end{array} \right) \end{array}$$

- *its initial state* c *is equal to* $\langle s, \emptyset \rangle$

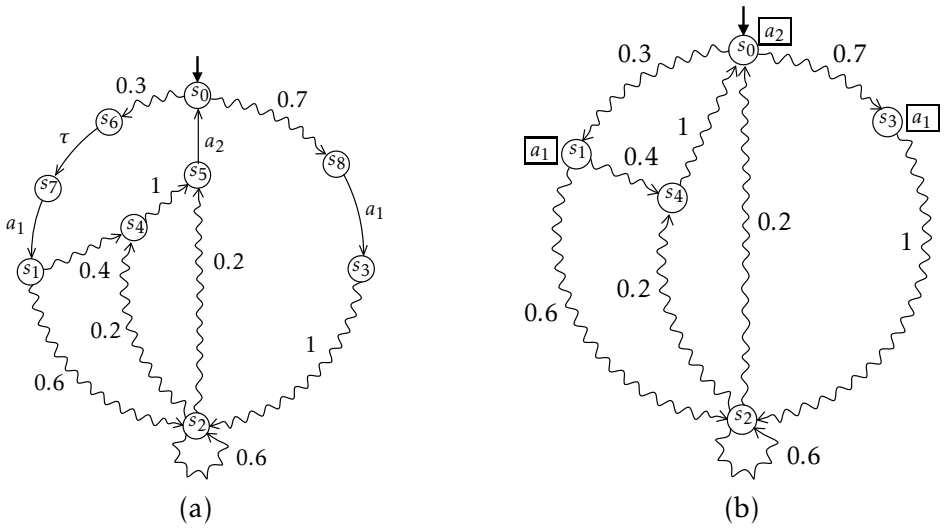


Figure 5.8: a DTAMC obtained from a tdIPC

– *the annotation function* $\mathscr{A} : C \mapsto \mathcal{P}^{\times}_{\leq k}(\mathcal{A})$ *is such that* $\left(\forall \langle s_i, \mathscr{S}\rangle \in C\right) \quad \mathscr{A}(\langle s_i, \mathscr{S}\rangle) = \mathscr{S}$.

Example 5.12. *For the urgency cut tdIPC P depicted in figure 5.6(c) an redrawn in figure 5.8(a), the associated DTAMC* $\mathcal{M}(P) = \langle S, \rightsquigarrow, s_0, \mathscr{A}\rangle$ *is depicted in figure 5.8(b). To study the distribution of the latency between action* a_1 *and action* a_2 *in P, we can use* $\mathcal{M}(P)$*. The distribution of this latency was studied in example 3.2 of chapter 3, setting two sets* α *and* ω*.* α *is the set of states in which the latency can be started,* $\alpha = \{s \in S \mid a_1 \in \mathscr{A}(s)\}$*.* ω *is the set of states in which the latency is ended,* $\omega = \{s \in S \mid a_2 \in \mathscr{A}(s)\}$.

Naturally, because a dIPC is also a tdIPC, a DTAMC can be associated to it. For any dIPC obtained by scheduling of a tdIPC, an interesting property is ensured: whatever the chosen scheduler $\mathfrak{s}$ to schedule a time deterministic IPC P, the DTAMC associated to P is the same as the DTAMC associated to the scheduled deterministic IPC $\mathfrak{s}(P)$.

Lemma 5.7. *Consider an IPC P and an arbitrary scheduler* $\mathfrak{s}$ *on P. The associated DTAMC to P,* $\mathcal{M}(P)$*, and the associated DTAMC to* $\mathfrak{s}(P)$*,* $\mathcal{M}(\mathfrak{s}(P))$*, satisfy:*

$$\mathcal{M}(P) \sim_p \mathcal{M}(\mathfrak{s}(P))$$

Proof. Consider a time deterministic IPC P. By definition of a time deterministic IPC (definition 5.11), we have that whatever is the followed path between an interactive state s and a probabilistic state s' of P, the multiset of taken actions is the same. The associated DTAMC to the time deterministic IPC P is just constructed over this multiset of actions, i.e., it does not take into account the different possible paths.

By scheduling of P with an arbitrary scheduler $\mathfrak{s}$, we just fix one of the possible paths between states s and s'. The associated DTAMC to this scheduled IPC is also constructed over the multiset of taken actions between s and s', and this multiset is the same in P and in $\mathfrak{s}(P)$. DTAMCs associated to P and to $\mathfrak{s}(P)$ are consequently the same, and thus they are strongly bisimilar. □

This property authorizes to consider dIPCs in place of tdIPCs. Although considering dIPCs is more restrictive than considering tdIPCs we can obtain the same underlying DTAMC models for performance evaluation.

5.2.5 Performance Measures Preservation

By using a compositional approach to model complex systems by IPCs, we ensured that, from end to end, the probabilistic branching bisimulation is preserved. Finally, if the IPC model is a tdIPC, we associate to it a DTAMC in the aim of extracting performance measures. This approach is valid only if we are able to ensure that associated DTAMCs of two branching probabilistic bisimilar tdIPCs allow to get the same result for a given performance result. In other words, two probabilistic branching bisimilar tdIPCs should yield two strongly bisimilar DTAMCs.

Lemma 5.8. *Associated DTAMCs of probabilistic branching bisimilar tdIPCs are strongly bisimilar, i.e.,*

$$\Big(\forall\,(P_1,P_2)\in\mathscr{P}^2\Big)\quad \Big(P_1\approx P_2\Big)\Longrightarrow\Big(\mathcal{M}(P_1)\sim_p\mathcal{M}(P_2)\Big)$$

Proof. See appendix C. □

5.2.6 Discrete-Time Annotated Markov Chain to Semi-Markov Chain

We define a second transformation from a DTAMC $\mathcal{M}(P)$ associated to an IPC P to an SMC that allows to compute the very same performance measures than obtained from $\mathcal{M}(P)$. In particular, we focus on SMCs for which the distributions associated to transitions (defining the sojourn time in states) are constant distributions, $\mathbf{C}_n:\mathbb{N}\mapsto[0,1]$, defined by

$$\Big(\forall n\in\mathbb{N}\Big)\quad \mathbf{C}_n(n)=1\wedge\Big(\Big(\forall n'\neq n\Big)\quad \mathbf{C}_n(n')=0\Big)$$

Definition 5.15 (SMC associated to a DTAMC)**.** *Consider a DTAMC $\mathcal{M}(P)=\langle C,\rightsquigarrow,c,\mathscr{A}\rangle$ associated to an urgency cut tdIPC $P=\langle S,\mathcal{A},\longrightarrow,\rightsquigarrow,s\rangle$. Let k be the cardinal of the largest annotation multiset associated to states (k is the length of the longest sequence of interactive transitions in P). An SMC $\mathcal{M}_S=\langle S,\mathcal{A},\ \square\!\!\rightsquigarrow,\mathcal{D},s_0\rangle$ associated to $\mathcal{M}(P)$ is such that:*

For every state $\langle s_2,\mathscr{S}_2\rangle\in C$ where $\mathscr{S}_2=\{a_1,\ldots,a_n\}$ ($n\leq k$ and $\Big(\forall i\leq n\Big)\ a_i\in\mathcal{A}$), and for each transition $\langle s_1,\mathscr{S}_1\rangle\overset{p}{\rightsquigarrow}\langle s_2,\mathscr{S}_2\rangle$ (for some $\langle s_1,\mathscr{S}_1\rangle\in C$ and $p\in\,]0,1]$), there is a sequence of transitions:

$$s_1\ \overset{\tau\ p}{\square\!\!\rightsquigarrow}\ s_1'\ \overset{a_1\ 1}{\square\!\!\rightsquigarrow}\ s_2'\ \overset{a_2\ 1}{\square\!\!\rightsquigarrow}\ \cdots\ \overset{a_{n-1}\ 1}{\square\!\!\rightsquigarrow}\ s_n'\ \overset{a_n\ 1}{\square\!\!\rightsquigarrow}\ s_2$$

in $\mathcal{M}_S$, satisfying:

- $\mathcal{D}\Big(s_n'\ \overset{a_n\ s}{\square\!\!\rightsquigarrow}\ {}_2\Big)=\mathbf{C}_1$
- $\Big(\forall i<n\Big)\quad \mathcal{D}\Big(s_i'\ \overset{a_i\ s}{\square\!\!\rightsquigarrow}\ {}'_{i+1}\Big)=\mathbf{C}_1$
- $\mathcal{D}\Big(s_1\ \overset{\tau\ p}{\square\!\!\rightsquigarrow}\ s_1'\Big)=\mathbf{C}_{k-n+1}$

The long run average vector of the SMC $\mathcal{M}_S$ associated to the DTAMC $\mathcal{M}(P)$ is strongly related to the steady state probability vector of the $\mathcal{M}(P)$.

Lemma 5.9 (Steady state probabilities in the associated SMC)**.** *Consider a DTAMC $\mathcal{M}(P)$ associated to an IPC P, $\mathcal{M}(P) = \langle C, \rightsquigarrow, c_0, \mathscr{A} \rangle$, and $\mathcal{M}_S(P) = \langle S, \mathcal{A}, \boxed{\rightsquigarrow}, \mathcal{D}, s_0 \rangle$ the SMC associated to $\mathcal{M}(P)$. Let k be the cardinal of the largest annotation multiset associated to states of $\mathcal{M}(P)$. We note π_c the steady state probability of a state $c \in C$ in $\mathcal{M}(P)$ and $\tilde{\pi}_s$ the steady state probability of a state $s \in S$ in $\mathcal{M}_S$.*

Consider a state $\langle s, \mathscr{S} \rangle \in C$ where $\mathscr{S}_2 = \{a_1, \ldots, a_n\}$ ($n \leq k$ and $\left(\forall i \leq n\right)\ a_i \in \mathcal{A}$), and $\mathscr{S}_{\rightsquigarrow \langle s, \mathscr{S} \rangle}$ the set of predecessors of $\langle s, \mathscr{S} \rangle$ in $\mathcal{M}(P)$. For every state $\langle s_i, \mathscr{S}_i \rangle \in \mathscr{S}_{\rightsquigarrow \langle s, \mathscr{S} \rangle}$ such that $\langle s_i, \mathscr{S}_i \rangle \overset{p_i}{\rightsquigarrow} \langle s, \mathscr{S} \rangle$, there is a sequence of transitions:

$$s_i \overset{\tau\ p_i}{\boxed{\rightsquigarrow}} s'_{i,1} \overset{a_1\ 1}{\boxed{\rightsquigarrow}} s'_{i,2} \overset{a_2\ 1}{\boxed{\rightsquigarrow}} \cdots \overset{a_{n-1}\ 1}{\boxed{\rightsquigarrow}} s'_{i,n} \overset{a_n\ 1}{\boxed{\rightsquigarrow}} s_2$$

in $\mathcal{M}_S$. Steady state probabilities of predecessors of s_2 in $\mathcal{M}_S$ are linked to the steady state probability of s_2 in $\mathcal{M}(P)$:

$$\left(\forall j < n\right) \quad \sum_{\langle s_i, \mathscr{S}_i \rangle \in \mathscr{S}_{\rightsquigarrow \langle s, \mathscr{S} \rangle}} \left(\tilde{\pi}_{s_{i,j}}\right) = \frac{\pi_{\langle s, \mathscr{S} \rangle}}{k}$$

Proof. See appendix D. □

5.3 Interactive Probabilistic Chains in Practice: Application to a FIFO Queue

We are going to use the IPC framework on the system presented in example 1.1 of chapter 1, i.e., a system composed of a producer, a FIFO queue and a consumer. We target the same performance results as in section 3.4, i.e., the pop latency distribution and the end-to-end latency distribution in the queue. Parameters of the system are set as in example 2.1 of chapter 2. As a reminder, those parameters are:

- the queue size, Q_S, set to 1,
- the push operation delay, D_{PUSH}, set to 2 time steps,
- the pop operation delay, D_{POP}, set to 1 time step,
- the production delay, D_{PROD}, set either to 2 time steps with a probability 0.5 or to 3 time steps with a probability 0.5,
- and the consumption delay, D_{CONS}, set either to 1 time step with a probability 0.5 or to 4 time steps with a probability 0.5.

We follow a hierarchical approach, modeling the whole system by several IPC subprocesses composed in parallel. The decomposition in subprocesses is rather natural: one process for the FIFO queue, one process for the producer and one process for the consumer. Let P_{FIFO}, P_{PROD} and P_{CONS} be the IPC models of respectively the FIFO queue, the producer and the consumer. To depict the behavior of the system, four actions are identified:

- The request for a push operation (pushRq): The producer initiates an insertion of element in the queue.
- The response for a push operation (pushRsp): The push operation was processed by the queue, the element is effectively inserted.
- The request for a pop operation (popRq): The consumer initiates the withdrawal of an element from the queue.
- The response for a pop operation (popRsp): The pop operation was processed by the queue, the element is effectively withdrawn and provided to the consumer.

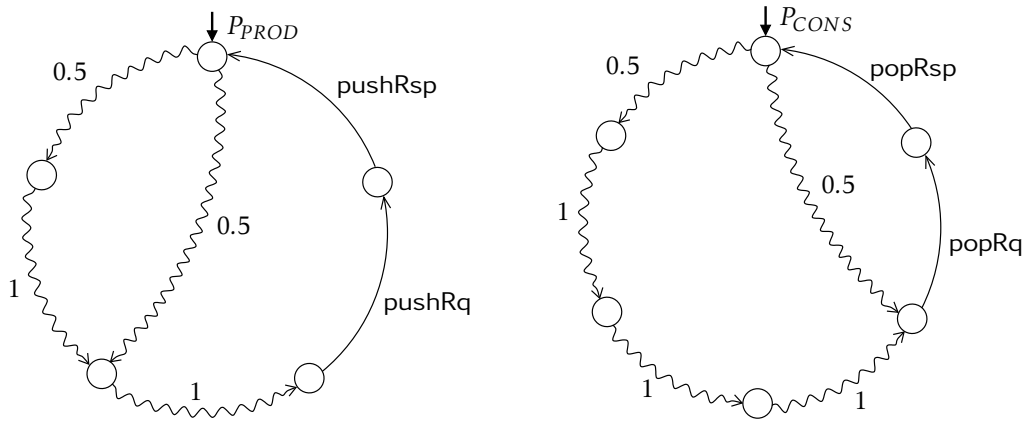


Figure 5.9: Producer and consumer IPC models

The whole system IPC model, P, is consequently specified by the parallel composition of the subprocesses P_{FIFO}, P_{PROD} and P_{CONS} with synchronization on actions pushRq, pushRsp, GRQ and popRsp:

$$\begin{aligned} P \;=\; & P_{PROD} \\ & |[\mathsf{pushRq}, \mathsf{pushRsp}]| \\ & P_{FIFO} \\ & |[\mathsf{popRq}, \mathsf{popRsp}]| \\ & P_{CONS} \end{aligned}$$

Producer and consumer behaviors are rather small (six states). Their IPC models P_{PROD} and P_{CONS} are depicted in figure 5.9. On the producer side, initially, no action can be fired: the time between two produced elements (D_{PROD}) has to elapse. At the end of this delay, the producer initiates a push operation (pushRq), waits for the signal of end of the operation (pushRsp), and loops. The consumer side is similar: initially the time between two consumed elements (D_{CONS}) has to elapse. At the end of this delay, the consumer initiates a pop operation (popRq), waits for the result (popRsp) and loops. The

The behavior of the studied FIFO queue is also not too complex, mainly because of its limited size (1 element) and because of the nature of the inserted delays. Nondeterminism is induced in the FIFO queue by the possibility to receive asynchronously either a push operation request or a pop operation request. Initially, the queue is empty, and it can only accept the insertion of an element. Then, after a first element insertion, the queue is full and it can only accept the withdrawal of the element. The IPC model, P_{FIFO} of the FIFO queue is depicted in figure 5.10.

The model of the whole system, P, has 49 states and 80 transitions. Because P is closed (there is no more interaction with the environment), urgency-cut can be applied. Let $P_{\not\to_{\mathcal{A}}} = \langle S, \mathcal{A}, \longrightarrow, \rightsquigarrow, s\rangle$ be the IPC obtained by urgency cut of P. We have $\mathcal{A} = \{\mathsf{pushRq}, \mathsf{pushRsp}, \mathsf{popRq}, \mathsf{popRsp}\}$. $P_{\not\to_{\mathcal{A}}}$ is depicted in figure 5.11. $P_{\not\to_{\mathcal{A}}}$ remains a large IPC (to be depicted) with 49 states and 62 transitions.

We can easily verify that $P_{\not\to_{\mathcal{A}}}$ is time deterministic. To compute performance measures, $P_{\not\to_{\mathcal{A}}}$ is transformed into a DTAMC as presented in section 5.2.4. Because we are targeting performance measures on the long run, we can limit our study to an irreducible part of the DTAMC, defining arbitrarily the initial state. The chosen irreducible DTAMC, called $\mathcal{M}(P_{\not\to_{\mathcal{A}}})$,

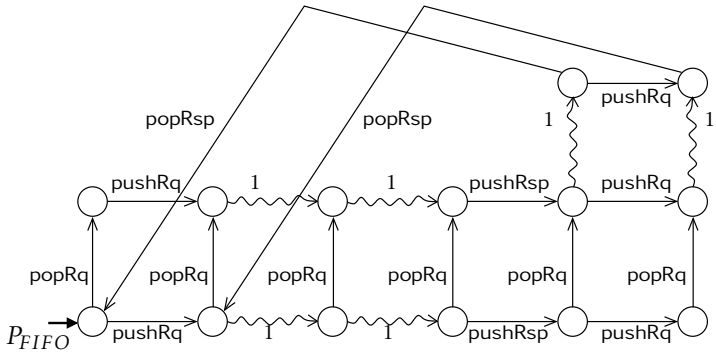
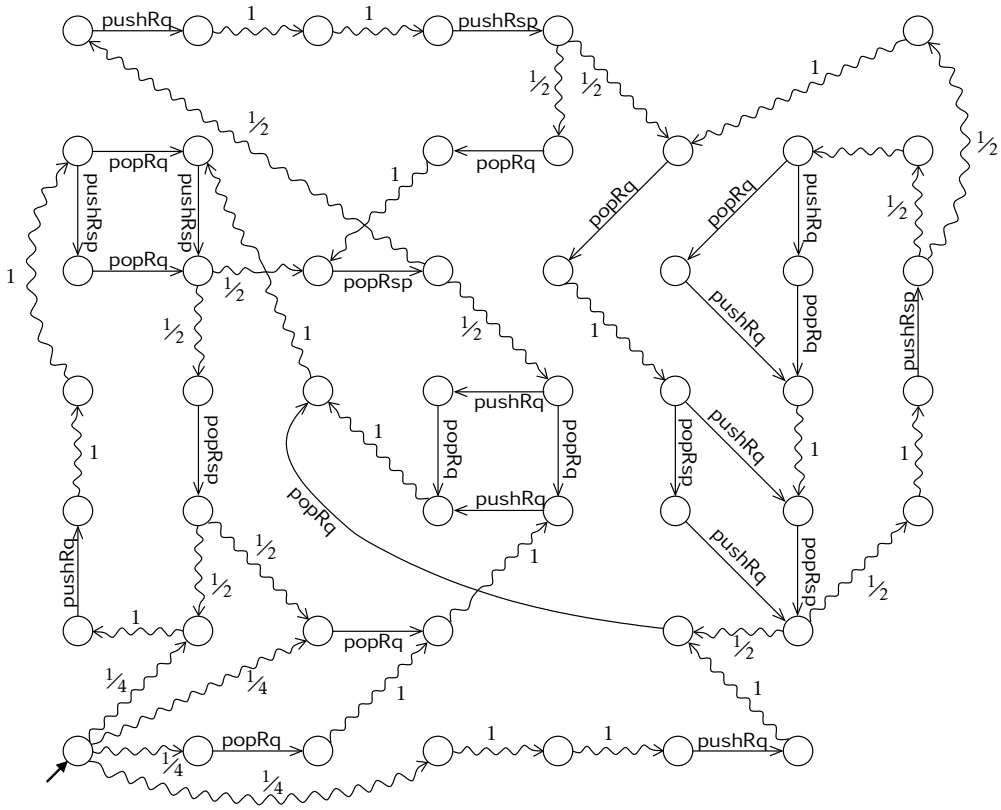


Figure 5.10: FIFO queue IPC model


pushRq
1
1
pushRsp
popRq
popRq
popRsp
pushRq
popRq
pushRq
popRq
popRsp
pushRq
popRq
pushRq
popRq
pushRq
popRq

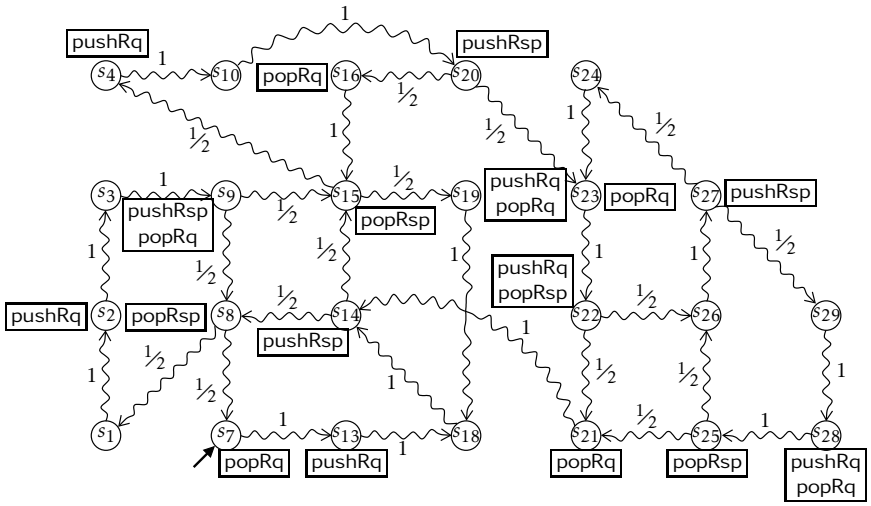


Figure 5.12: Whole system irreducible DTAMC model

is depicted in figure 5.12. The set of states of $\mathcal{M}(P_{\not\to\not\to_A})$ is the same for all irreducible DTAMC we could have chosen. The DTAMC $\mathcal{M}(P_{\not\to\not\to_A})$ can now be used to compute the latency distribution of the pop operation and the latency distribution of the end-to-end latency in the queue.

For the pop operation latency, information concerning the pop operation (i.e., popRq and popRsp actions) are useful. We can thus consider the DTAMC $\mathcal{M}'(P_{\not\to\not\to_A})$ for which the annotation function ignores actions pushRq and popRq. $\mathcal{M}'(P_{\not\to\not\to_A})$ is not minimal. The computation of the pop operation latency distribution can be processed on its quotient $\mathcal{M}'(P_{\not\to\not\to_A})_{/_\approx}$, which is the DTAMC already studied in section 3.4.1 and depicted in figure 3.4.

We could have followed another methodology to get the same DTAMC as $\mathcal{M}'(P_{\not\to\not\to_A})_{/_\approx}$. Because only popRq and popRsp actions are useful for the pop operation latency distribution computation, the IPC $P_{\not\to\not\to_A}$ can be first minimized (according to the IPC probabilistic branching bisimulation), hiding pushRq and pushRsp actions. Then this reduced IPC can be transformed into a DTAMC, which is the same as $\mathcal{M}'(P_{\not\to\not\to_A})_{/_\approx}$.

For the end-to-end latency, the DTAMC $\mathcal{M}(P_{\not\to\not\to_A})$ cannot be directly used. Indeed, the queue size information for each state is needed. This information can be easily added to the DTAMC: the graph is explored from the initial state for which the queue size is equal to zero. At each pushRsp action, the queue size is incremented by one, and at each popRsp the queue size is decremented by one. Then, the DTAMC $\mathcal{M}''(P_{\not\to\not\to_A})$ for which the annotation function is limited to pushRq, popRsp actions and the queue size can be used to compute the end-to-end latency distribution. $\mathcal{M}''(P_{\not\to\not\to_A})$ is the DTAMC already studied in section 3.4.2 and depicted in figure 3.6.

5.4 Relating Interactive Probabilistic Chains to Interactive Markov Chains

From a methodology point of view, IPCs and IMCs present strong similarities:

– Both IMCs and IPCs aim at integrating labeled transitions systems and Markov chains in a single formalism

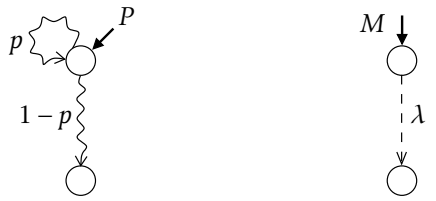


Figure 5.13: Approximation of an IMC by an IPC

- timed systems can be modeled following a compositional approach (with intermediate minimization according to bisimulations)
- a large set of delays can be taken into account thanks to approximations using the phase-type distributions
- the performance analysis of IMCs and IPCs relies on the study of an underlying Markov chain.

Nevertheless IMC and IPC semantics are very different. In IPC, because of the discrete synchronous characteristic of time, time has to progress synchronously for parallel composition or nondeterministic choices, which is expressed by the SOS semantic rules (4.b) and (5.b).

Although IPCs target the modeling of discrete-time systems, one can imagine to model continuous time systems by discretization of the time scale, as in [ZN10]: continuous time is divided in time slices representing the time unit in the IPC, i.e., the duration of a probabilistic transition. The quality of the discretization can be always improved by increasing the sampling rate, i.e., reducing the duration of a time slice. For instance, it is possible to model an IMC by an IPC. We illustrate this possibility by approximating an exponential distribution, which governs the sojourn time in states of an IMC, by a geometrical one, which governs the sojourn time in states of an IPC. For instance consider the two processes M and P depicted in figure 5.13 (Those processes are phase-type distributions corresponding to exponential and geometrical distributions and are also IMC and IPC processes).

Suppose that the time unit considered for the IMC model M is U. The sojourn time SJ_M in the initial state of M is λ-exponentially distributed. The expected value of SJ_M is $E[\mathsf{SJ}_M] = \frac{1}{\lambda}U$. We approximate M by the IPC P where the considered time unit (i.e., the duration of a probabilistic transition) is u. the sojourn time SJ_P in the initial state of P is geometrically distributed (with parameter p). The expected value of SJ_P is $E[\mathsf{SJ}_P] = \frac{1}{1-p}u$. We adjust the probability p such as to have

$$E[\mathsf{SJ}_M] = E[\mathsf{SJ}_P]$$

i.e., the exponential and the geometrical distributions are, on average, equal. The resolution of this equation gives us $p = 1 - \lambda\frac{u}{U}$, which is only correct if $\lambda\frac{u}{U} \leq 1$ (p is a probability and must satisfy $0 \leq p \leq 1$). the case $\lambda\frac{u}{U} > 1$ corresponds to a discretization of the continuous time scale of M, using unsuitable discrete time slices for P. For instance suppose $u = U$ and $\lambda = 5$. We have $\mathsf{SJ}_M = 0.2U$, which is difficult to approximate using discrete time slices of $1u = 1U$. The best approximation we can have is to consider the borderline case $p = 0$, which gives us $\mathsf{SJ}_P = 1$. The value 1 is the smallest possible sojourn time $\mathsf{SJ}_P = 1$ we can have under the hypothesis $\frac{u}{U} = 1$, i.e., $u = U$.

In figure 5.14, we compare the continuous-time distribution of SJ_M to the discrete-time distribution of SJ_P when approximating M by P for different values of u. In this example, λ is set to 0.4 and the probability p is set to $1 - \lambda\frac{u}{U}$. One can see that the smaller the time slice is,

the better the approximation is.

According to those observations, we can approximate, by discretization of the continuous-time exponential distributions, an IMC by an IPC. Moreover, the approximation can be made always better by increasing the sampling rate of the discretization, i.e., by making the time slice duration u tend to zero.

Because IMC and IPC semantics are rather different for parallel composition and non-deterministic choice (in the case where time is involved), it is interesting to study if the limit, when the time duration u of a probabilistic transition tends to zero, of the behavior of an IPC approximating an IMC is also preserved when using operators defined in the semantics. We only study the parallel composition operator, the following being adaptable for others operators, in particular the nondeterministic choice operator.

Consider the two IMCs M_1 and M_2 depicted on figure 5.15. The IMC M, also depicted on figure 5.15 is the parallel composition (with no synchronization) of M_1 and M_2, "$M = M_1 \,|[\varnothing]|\, M_2$". The considered time unit in those IMCs is U.

As we have seen, we approximate M_1 and M_2 by the two IPCs P_1 and IPC_2 depicted in figure 5.16. For P_1 and P_2, the time unit is u (it is the time duration of a probabilistic transition). To approximate M_1 by P_1 and M_2 by P_2, the probabilities p and q are defined as following:

$$p = 1 - \lambda\frac{u}{U} \qquad \text{and} \qquad q = 1 - \mu\frac{u}{U}$$

The IPC P, depicted in figure 5.16, is the parallel composition of P_1 and P_2, "$P = P_1 \,|[\varnothing]|\, P_2$". We study the behavior of the state s_0 of P when the time unit u tends to zero, and we compare to the behavior of the state s_0 of M. We introduce some notations first: $\mathsf{SJ}_M(s_0)$ and $\mathsf{SJ}_P(s_0)$ are the sojourn time of states s_0 in M and P. $\mathsf{Pr}_{[P_1]}$ (resp. $\mathsf{Pr}_{[M_1]}$) denotes the probability that the process P_1 (resp. M_1) leaves its initial state before P_2 (resp. M_2). Symmetrically, $\mathsf{Pr}_{[P_2]}$ (resp. $\mathsf{Pr}_{[M_2]}$) denotes the probability that the process P_2 (resp. M_2) leaves its initial state before P_1 (resp. M_1). Finally $\mathsf{Pr}_{[P_1\|P_2]}$ (resp. $\mathsf{Pr}_{[M_1\|M_2]}$) denotes the probability that the processes P_1 and P_2 (resp. M_1 and M_2) leave their initial state simultaneously.

To study behaviors in states s_0 in P and M when u tends to zero, we compare: $\mathsf{SJ}_M(s_0)$ to $\mathsf{SJ}_P(s_0)$, $\mathsf{Pr}_{[M_1]}$ to $\mathsf{Pr}_{[P_1]}$, $\mathsf{Pr}_{[M_2]}$ to $\mathsf{Pr}_{[P_2]}$ and $\mathsf{Pr}_{[M_1\|M_2]}$ to $\mathsf{Pr}_{[P_1\|P_2]}$.

In the IMC M the following results hold: $\mathsf{SJ}_M(s_0)$ is exponentially distributed with parameter $(\lambda + \mu)$, and

$$\mathsf{Pr}_{[M_1]} = \frac{\lambda}{\lambda+\mu} \qquad \mathsf{Pr}_{[M_2]} = \frac{\mu}{\lambda+\mu} \qquad \mathsf{Pr}_{[M_1\|M_2]} = 0$$

We first study the probabilities of probabilistic transitions of P (from state s_0) by asymptotic analysis. When u tends to zero (i.e., when $\frac{u}{U}$ tends to zero), we have:

$$pq = 1 - (\lambda+\mu)\frac{u}{U} + o\left(\frac{u}{U}\right) \qquad 1-p = \lambda\frac{u}{U} \qquad 1-q = \mu\frac{u}{U} \qquad (1-p)(1-q) = o\left(\frac{u}{U}\right)$$

One can first say that the sojourn time $\mathsf{SJ}_P(s_0)$ in state s_0 of P is geometrically distributed with parameter $pq = 1 - (\lambda+\mu)r + o(r)$. We saw that this geometric distribution tends to the exponential distribution with parameter $\lambda+\mu$ when u (and thus r) tends to zero. Consequently, when u tends to 0, the distribution of $\mathsf{SJ}_P(s_0)$ converges to the distribution of $\mathsf{SJ}_M(s_0)$.

Then, one can study $\mathsf{Pr}_{[P_1]}$, $\mathsf{Pr}_{[P_2]}$, and $\mathsf{Pr}_{[P_1\|M_2]}$. Those probabilities are conditional probabilities. For instance, $\mathsf{Pr}_{[P_1]}$ corresponds to the probability that the transition $s_0 \overset{1-p}{\rightsquigarrow} s_1$ is taken

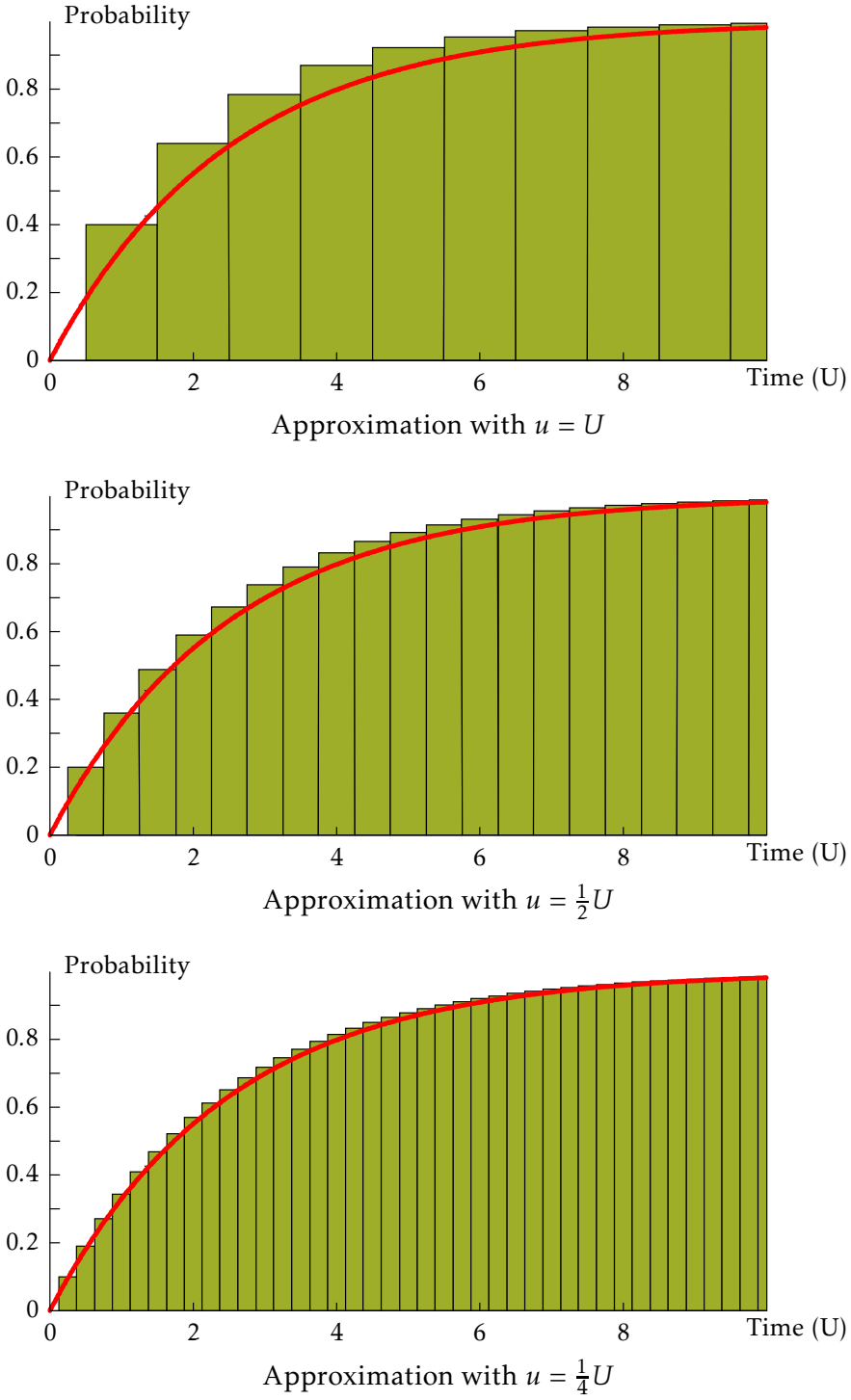


Figure 5.14: Distributions of SJ_M and SJ_P for different values of u

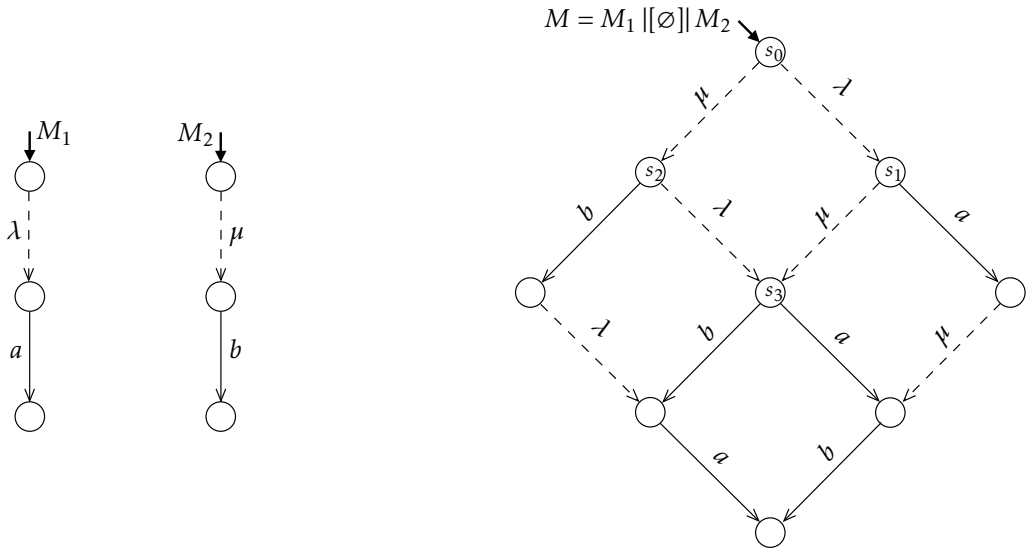


Figure 5.15: The parallel composition of two IMCs that let time progress initially

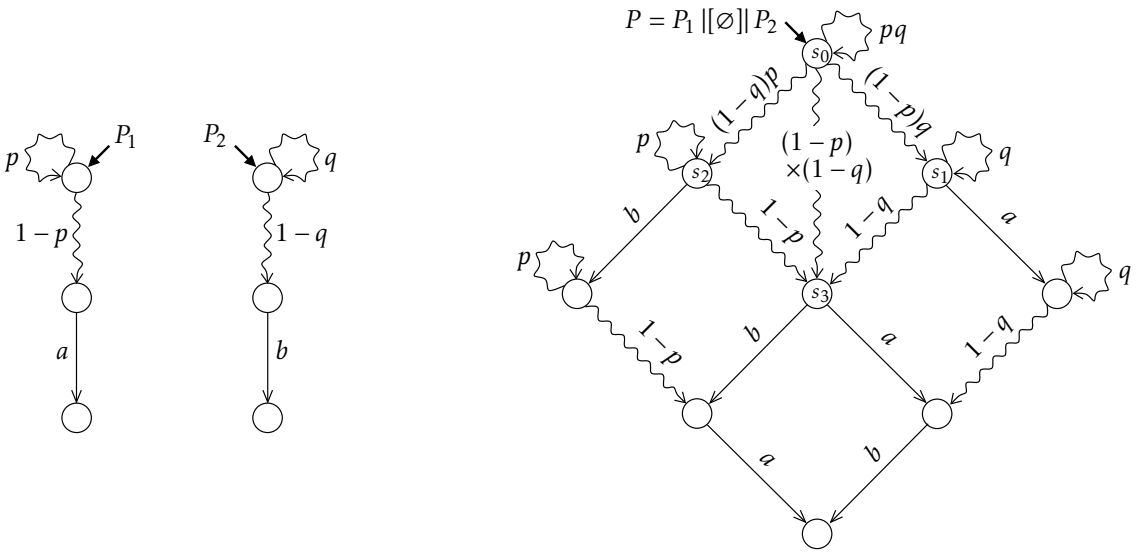


Figure 5.16: The parallel composition of two IPCs that let time progress initially

given that we leave the state s_0. Thus, we have simply

$$\mathsf{Pr}_{[P_1]} = \frac{1-p}{(1-p)+(1-q)+(1-p)(1-q)}$$

Following a similar reasoning, we have

$$\mathsf{Pr}_{[P_2]} = \frac{1-q}{(1-p)+(1-q)+(1-p)(1-q)} \quad \text{and} \quad \mathsf{Pr}_{[P_1 \| P_2]} = \frac{(1-p)(1-q)}{(1-p)+(1-q)+(1-p)(1-q)}$$

When u tends to zero, we have

$$\mathsf{Pr}_{[P_1]} \underset{u \to 0}{\longrightarrow} \frac{\lambda}{\lambda+\mu} \qquad \mathsf{Pr}_{[P_2]} \underset{u \to 0}{\longrightarrow} \frac{\mu}{\lambda+\mu} \qquad \mathsf{Pr}_{[P_1 \| P_2]} \underset{u \to 0}{\longrightarrow} 0$$

We can conclude that the limit of the behavior of an IPC approximating an IMC is preserved when using the parallel composition operator. On the same line, the limit of the behavior of an IPC approximating an IMC is preserved when using the nondeterministic choice operator.

Consequently, there exists strong links between IMC and IPC semantics, despite they can seem to be rather different at first sight. Nevertheless, there is a fundamental difference between an IMC and its approximation by an IPC: The approximation may be as accurate as wanted, there is always a non-zero probability that concurrent delays in IPCs may elapse simultaneously. In IMCs, concurrent delays can never elapse at the same time. On the previous example, for the IPC P, we had $\mathsf{Pr}_{[P_1 \| P_2]}$ that tends to zero (but is never equal to zero) when the approximation is improved. For the IMC M, we had $\mathsf{Pr}_{[P_1 \| P_2]}$ exactly equal to zero. The delays represented by M_1 and M_2 can never elapse simultaneously.

5.5 Discussion

In this chapter, we introduced interactive probabilistic chains, a formalism dedicated to model discrete-time systems in a compositional way. We defined both an operational semantics and strong and branching bisimulations for IPCs, which we showed to be congruences for the parallel composition. The congruence property ensures that intermediate minimization can be applied during compositions, which is required to make the compositional approach effective. The IPC language IPC_L has been simplified to be presented, but could be lifted to value passing rendez-vous (and local state variables) in LOTOS-style. This is used in our tool chain.

We also defined a transformation of an IPC into a DTAMC to exploit the performance methodologies we presented in chapters 2 and 3. This transformation proceeds in two steps: Firstly the IPC is transformed into an alternating model [Han94] (there are no more states presenting both nondeterministic and probabilistic choices). The transformation of an IPC into an alternating model relies on classical notions of maximal progress and urgency defined for timed models [NS91]. Secondly, the alternating model is transformed into a DTAMC.

This transformation of an IPC into a DTAMC imposes some restrictions concerning the use of nondeterminism in models: only fully specified models, i.e., models with no nondeterminism due to implementation freedom or external environment, can be tackled by this transformation. Although these restrictions reduce the applicability of methodologies presented in the previous chapters, we believe that the study of partially specified hardware systems does not provide enough exploitable results, as we illustrated in example 5.7.

We defined a second transformation of a DTAMC into an SMC. Computing latency distributions for the SMC associated to a DTAMC would require to adapt the methodology presented in chapter 3 to deal with SMCs instead of DTAMCs. Unfortunately, the methodology cannot be fully adapted to SMCs. For instance, we would not be able to compute the end-to-end latency distribution in section 5.3. Indeed, to compute this distribution, we have to consider, in addition to actions the system may fire, the size of the queue in the different states, which is not possible with SMCs since we are not able to store information concerning states within the SMC formalism. Although the SMC transformation may seem to be less complete (and thus adequate) than the DTAMC one, there is no information associated to states, which is an interesting property for implementation as we will see in the next chapter.

Although their semantics are rather different, IPCs and IMCs present strong similarities. In particular, the limit of an IPC when the considered time step tends to zero is an IMC. Nevertheless, there is a fundamental difference between IMCs and IPCs: contrary to IMCs, there is always a non null probability in IPCs that sojourn times in concurrent states elapse simultaneously.

Chapter 6

Implementation

In this chapter, we present the toolchain supporting the IPC formalism in practice. This tool chain is depicted in figure 6.1. The chain can be split in two parts: a first part dedicated to IPC modeling in a compositional approach, and a second part concerning performance evaluation using IPCs, mainly the computation of latency distributions. Some of the tools existed, as part of the "Construction and Analysis of Distributed Processes" (CADP) toolbox [GLMS07], while others have been developed.

We present guidelines for the use of this performance tool chain. We underline some problems encountered when using the IPC formalism and how we tackled them in practice. For developed tools, we focus on some interesting algorithmic aspects that could help to understand, in practice, the IPC formalism. Firstly, we present the way to model systems by IPCs. Secondly, we present the tools dedicated to performance evaluation, and mainly to the computation of latency distributions.

6.1 Interactive Probabilistic Chain Modeling

Originally designed for verifying the functional correctness of LOTOS specifications, the CADP toolbox has been enriched in the last years to support the IMC formalism [GH02, HJ03]. Thus, it allows to perform both functional verification and performance evaluation of distributed systems.

We followed a similar approach as the one used to support IMCs in the CADP toolbox: time information is introduced in a functionally verified LOTOS specification. While time is modeled by continuous phase-type distributions in IMCs, discrete phase-type distributions are used for IPCs.

6.1.1 Storing an IPC Using the CADP Toolbox

The CADP toolbox provides the BCG format to store LTS. In the BCG format, a label, stored as a character string, is associated to each transition. Because labels are stored as strings, the BCG format can be used not only to store LTS, but also other directed graphs like IMCs, the discrimination between different types of transitions appearing in the transition labels. Consequently, there are some conventions defined in the CADP toolbox concerning how different

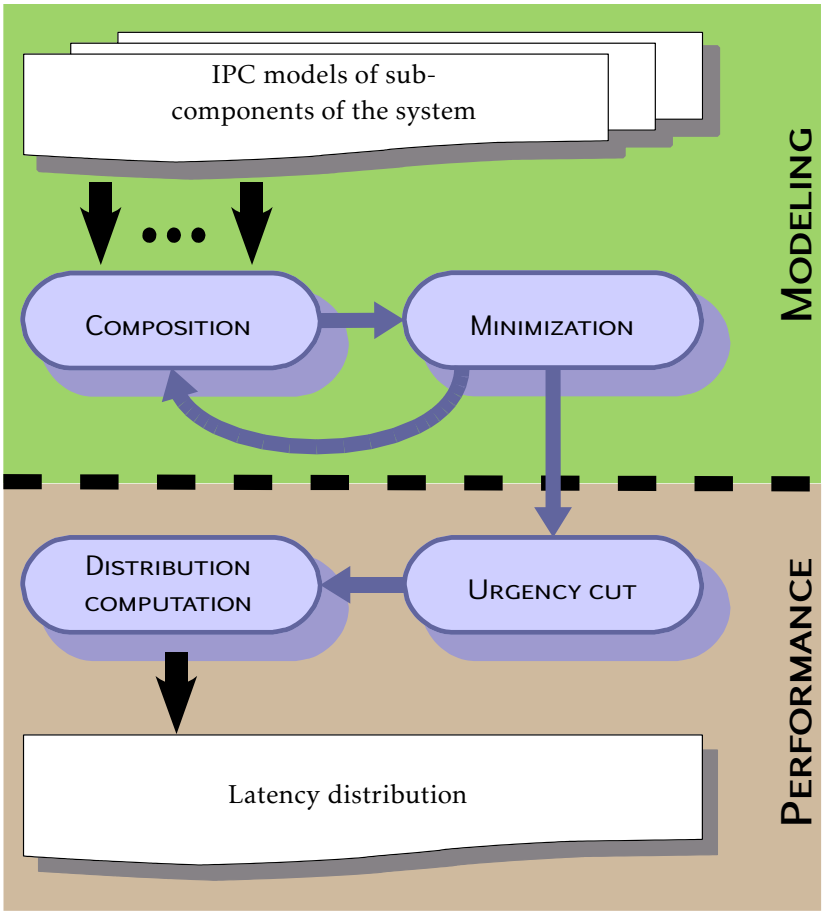


Figure 6.1: Developed tool chain

types of transitions have to be encoded. To distinguish the different transitions of IPCs in a Bcg graph, we reuse the following conventions:

- "ACT" is the string associated to an interactive transition labeled with the action ACT.
- "prob p" is the string associated to a probabilistic transition with a probability p, p being a real value greater than zero and less or equal to 1.

Cadp includes an Application Programming Interface (API) dedicated to the manipulation of Bcg graphs. The Bcg format is the input format used by all tools developed (using the Bcg API) to deal with IPCs.

6.1.2 Using Lotos to Express Interactive Probabilistic Chains

Due to strong similarities between the semantics of Lotos and IMCs, it is possible to use directly the Lotos language to express IMCs [GH02]. Indeed, Markovian transitions in IMCs are simply interleaved nondeterministically with interactive transitions. Consequently, a two step approach can be followed. Firstly, a Lotos specification including special actions Λ_i, which must not be used for synchronization, to express Markovian delays can be directly written and then compiled [Gar89] with the Caesar.adt and Caesar compilers. Then, the Labeled Transition System (LTS) generated by Caesar can be transformed into an IMC by renaming transitions labeled with gates Λ into Markovian transitions of the form "rate λ".

In order to reuse existing tools and existing functional LOTOS models, we would like to apply the same kind of two step approach for IPCs. However, an extension of the Lotos algebra with constructs supporting IPCs would imply a large effort in the modification of LTS generation tools. Indeed, contrary to the expression of IMCs using Lotos, the time model of IPCs does not allow to use directly the Lotos language. Thus, some restrictions in the use of Lotos operators are mandatory to generate IPCs. To denote a probabilistic transition in the Lotos specification, we use special actions Π_i that are not allowed to be synchronized. In an IPC context, the Lotos operators are used as following:

- Sequential composition, hiding and process instantiation operators can be used as for Lotos specifications. Indeed, for those operators, the SOS rules of IPCs are compliant with the rules of Lotos process algebra.
- The probabilistic choice operator can be written using the nondeterministic choice operator. The probabilistic choice is written as the nondeterministic choice between several Π actions. When those Π actions are instantiated by probabilities in the LTS, the user has to ensure that all those probabilistic transitions sum up to one.
- Nondeterministic choice between actions is allowed as in Lotos. In addition, we can define a nondeterministic choice between actions and one single probabilistic choice, the semantics rules of Lotos and IPC being, in this case, compliant. On the contrary, a nondeterministic choice between several probabilistic choices cannot be expressed using Lotos.
- Parallel composition is not allowed: the Lotos and IPC semantics are different.

When looking at this methodology to use Lotos to generate IPCs, we can see that there are a lot of restrictions that prevent us to write complex IPCs. Mainly, the parallel composition is missing, although it is mandatory for a compositional approach. Nevertheless, this methodology enables us to reuse Lotos descriptions of functional models, resulting in a significant time gain.

To generate large IPCs, in spite of those limitations, Lotos descriptions are used to gen-

erate quickly and efficiently small sequential subcomponents of the system. Thanks to the implementation of the parallel composition at the IPC level, complex IPCs are obtained by composition of the simple subcomponents.

6.1.3 Compositional Approach to Generate Interactive Probabilistic Chains

LOTOS allows to write sequential IPCs simply. But this implies that delays, modeled as discrete phase-type distributions, are directly incorporated in LOTOS models.

Firstly, this task prevents us from a modular approach: we cannot have a clear distinction between the functional part and the timed part of the specification. In particular, one may be interested in having a functional specification of the whole system, which is enriched, in a second step, with time information.

Secondly, it prevents us from following a compositional approach, because we cannot use the parallel composition operator in LOTOS models. It implies that it will be difficult, not to say infeasible, to model complex systems by IPCs using LOTOS.

As a consequence, we do not use LOTOS to model a whole complex system, but rather apply the following approach:

- functional subcomponents (i.e., without time information) are modeled in LOTOS
- functional LTS models of the subcomponents are generated using the CAESAR compiler.
- functional LTS models of the subcomponents are enriched with time information, using the tool IPC_INSERT to get IPC models of subcomponents
- IPC models of the subcomponents are composed in parallel (according to the parallel composition semantics of IPC) using a dedicated tool, IPC_COMPOSE

One can see that this approach does not prevent us from performing functional verification since functional models of subcomponents can be composed (according to the LTS semantics) to get a functional model of the whole system.

In the rest of this section, we first discuss the enrichment of functional models with time information and we motivate why the insertion of time information into functional models required the development of a dedicated tool IPC_INSERT. Then, we present the tool IPC_COMPOSE that implements the parallel composition (with synchronizations) for IPCs.

6.1.4 Constraint-Oriented Specification of Interactive Probabilistic Chains

In this section, we discuss the issues related to the constraint-oriented specification of IPCs as proposed for IMCs [HK00]. We would like to clearly separate time information from functional specification, inserting time information by composition. Delays, modeled by discrete phase-type distributions in separated processes, are seen as time constraints and composed in parallel with the functional specification. This methodology implies that actions denoting the beginning and the end of the delay associated to a temporized operation appear explicitly in functional models. We call those actions *begin-action* and *end-action*, and we note them a_b and a_e. The new functional model, compliant with a constraint-oriented insertion of delays, should remain branching equivalent to the initial one, after hiding begin and end actions.

For the constraint-oriented specification of IPCs, we distinguish two approaches, one using the standard parallel composition operator of IPCs, and one using a modified composition operator. Whilst the former approach may lead to wrong results, we show that the latter approach

always gives us the expected results.

Inserting Delays Using the Standard Parallel Composition Operator

To specify IPCs in a constraint-oriented way, a first idea would be to use the standard parallel composition operator (as for IMCs). We illustrate this approach in the following example.

Example 6.1. *Consider a system that executes a temporized operation. In a functional view, the operation is usually considered as non atomic and is modeled by two atomic actions: the beginning of the operation and the end of the operation. By using notations a_b and a_e for the beginning and the end of the temporized operation, the functional behavior of this system is simply*

$$B_f = a_b \,;\, a_e \,;\, \delta$$

which means that action a_e is taken after action a_b. Suppose that the operation takes one time unit to be processed, i.e., the action a_e follows action a_b after exactly one time unit. Time can be directly taken into account by modeling the system by the behavior,

$$B_t = a_b \,;\, \sum_{\bullet} 1 :: (a_e \,;\, \delta)$$

B_t can also be obtained by insertion of a delay of one time unit in a constraint-oriented way. Actions a_b and a_e identify the beginning and the end of the delay and can be used to insert the delay compositionally. The delay can be modeled by the IPC behavior

$$B_d = a_b \,;\, \sum_{\bullet} 1 :: (a_e \,;\, B_d)$$

which means that, between the beginning and the end of the delay, one time unit elapses.

The timed behavior B'_t of the system is finally obtained by parallel composition of B_f and B_d,

$$B'_t = B_f \,|[a_b, a_e]|\, B_d$$

The process B'_t, obtained in a compositional way, is strongly bisimilar to the targeted process B_t.

The parallel composition seems to be an efficient means to insert delays in functional models. Unfortunately for IPCs, in some cases, the use of the parallel composition for delays insertion may lead to unexpected results. This problem is not encountered when modeling systems with IMCs. We illustrate this problem in the two following examples where a same system is modeled both using IMCs and IPCs. In the first example, the insertion of delay using the IPC parallel composition does not induce nondeterminism whilst it does in the second example.

Example 6.2. *Consider a system where an action a_1 may be taken if and only if a delay Δ has not yet elapsed, and where an action a_2 may be taken if and only if the delay Δ has elapsed.*

We model this system both as an IMC and as an IPC, inserting the delay Δ by using the parallel composition operators of IMCs and IPCs. The functional model is the same for IMC and IPC and is depicted in figure 6.2, where actions a_b and a_e denote the begin and end actions used for the insertion of delay Δ.

In the case of the IMC model, we suppose that the delay Δ is modeled by an exponential distribution with rate λ. In the case of the IPC model, the delay Δ is modeled by a constant distribution of

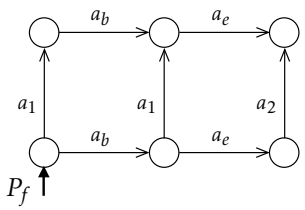


Figure 6.2: Functional model of the system. Actions a_b and a_e denote the begin and end of delay Δ

two time steps. Those delays are modeled by an IMC model, M_d, and an IPC model, P_d, depicted in figures 6.3(a) and (b).

The expected IMC model, M_t, and IPC model, P_t, of the timed system are depicted in figure 6.3(c) and in figure 6.3(d), respectively.

We try to construct models M_t and P_t, by constraint-oriented insertion of the delays, using IMC and IPC parallel composition operators. After hiding a_b and a_e actions, and minimization (with respect to the branching bisimulations of IMCs and IPCs), the models obtained are:

- *IMC model:* $M_{t'} = \left(\mathsf{hide}\ (a_b, a_e)\ \mathsf{in}\ P_f\ |[a_b, a_e]|\ M_d\right)_{/\approx}$
- *IPC model:* $P_{t'} = \left(\mathsf{hide}\ (a_b, a_e)\ \mathsf{in}\ P_f\ |[a_b, a_e]|\ P_d\right)_{/\approx}$

$M_{t'}$ and $P_{t'}$ are depicted in figure 6.3(e) and (f).

We can see that in both cases, the models obtained by constraint-oriented insertion of delays are not branching bisimilar to the expected one, i.e., $M_{t'} \not\approx M_t$ and $P_{t'} \not\approx P_t$. However, we can note that:

- *the expected IMC model M_t is Markovian branching bisimilar to the model $M_{t'}$ modulo urgency cut, i.e., $M_{t\not\to_{\mathcal{A}}} \approx M_{t'\not\to_{\mathcal{A}}}$*
- *the expected IPC model IPC_t is probabilistic branching bisimilar to the model $IPC_{t'}$ modulo urgency cut, i.e., $P_{t\not\to_{\mathcal{A}}} \approx P_{t'\not\to_{\mathcal{A}}}$*

The urgency cut IMC $M_{t\not\to_{\mathcal{A}}}$ and the urgency cut IPC $P_{t\not\to_{\mathcal{A}}}$ are depicted in figure 6.3(g) and (h).

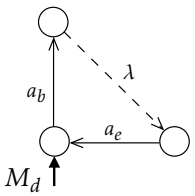
For both IMC and IPC, the constraint-oriented insertion of delays introduces nondeterminism in the models. However, when the constraint-oriented insertion is processed on the functional model of a closed system, urgency-cut can be applied, leading to the expected IMC or IPC model.

We are also interested in the constraint-oriented insertion of delays in non-closed models, i.e., in subcomponents of a closed system we want to compose. In other words, in IMC and IPC modeling, one may want to use a subcomponent obtained by constraint-oriented insertion of delays using the parallel composition, in place of the model directly written with delays.

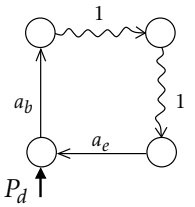
In subcomponent models, the constraint-oriented insertion of delays introduces nondeterminism we cannot avoid (urgency cut cannot be applied because models are not closed). Consequently, one has to study the impact of the introduced nondeterminism on the composition of non-closed models.

We illustrate the construction of a closed system, by using timed subcomponent models obtained in a constraint-oriented way in the following example.

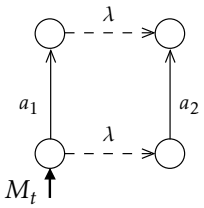
Example 6.3. *Consider the same system as in example 6.2, modeled using both IMCs and IPCs. Suppose that this system is not closed, i.e, it is a subcomponent C_1 of a closed system S. On may want to use the IMC model $M_{t'}$ of the subcomponent C_1 in place of the IMC model M_t, and the IPC*



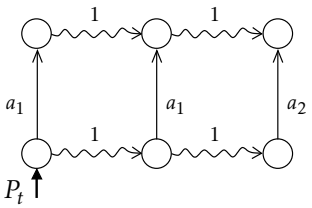
(a) IMC delay model



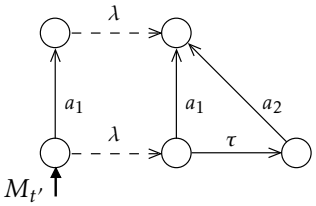
(b) IPC delay model



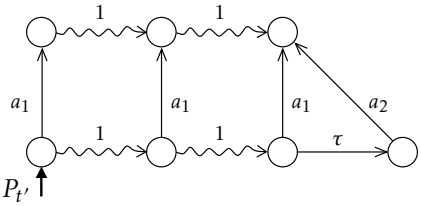
(c) Expected IMC model



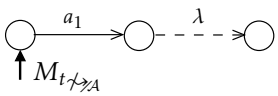
(d) Expected IPC model



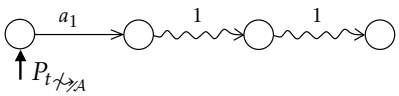
(e) IMC model:
$\left(\text{hide } (a_b, a_e) \text{ in } P_f \,|[a_b, a_e]|\, M_d\right)_{/\approx}$



(f) IPC model:
$\left(\text{hide } (a_b, a_e) \text{ in } P_f \,|[a_b, a_e]|\, P_d\right)_{/\approx}$



(g) Urgency cut expected IMC



(h) Urgency cut expected IPC

Figure 6.3: Insertion of delays in IMCs and in IPCs using parallel composition

model $P_{t'}$ of the subcomponent C_1 in place of the IPC model P_t to construct the model of the targeted closed system S. As we noticed previously, the construction of $M_{t'}$ and $P_{t'}$ introduces unexpected nondeterminism.

For instance, consider that the closed system S is the parallel composition of the subcomponent C_1 synchronized with a second subcomponent C_2 on actions a_1 and a_2. The subcomponent C_2 behaves as following: it can choose nondeterministically between actions a_1 and a_2 after a delay Δ' is elapsed. If the action a_1 is taken, then action a_3 may be taken. Otherwise, action a_2 is taken, possibly followed by a'_3.

In the case of the IMC model, we suppose that the delay Δ' is modeled by an exponential distribution with rate μ. In the case of the IPC model, the delay Δ' is modeled by a constant distribution of two time steps. The IMC model $M_{t''}$ (cf. figure 6.4(a)) and the IPC model $P_{t''}$ (cf. figure 6.4(b)) of the subcomponent C_2 are written with direct insertion of the delay Δ'.

The expected closed system is consequently modeled by:
- *the IMC $M = M_{t''} \,|[a_1, a_2]|\, M_t$*
- *the IPC $P = P_{t''} \,|[a_1, a_2]|\, P_t$*

Because M and P model a closed system, urgency cut can be applied. The urgency cut IMC $M_{\not\rightarrow_A}$ of M and the urgency cut IPC $P_{\not\rightarrow_A}$ of P are depicted in figure 6.4(c) and (d).

Consider the models M' and P' respectively obtained by replacing M_t and P_t by $M_{t'}$ and $P_{t'}$. Because M' and P' model a closed system, urgency cut can be applied. The urgency cut IMC $M'_{\not\rightarrow_A}$ of M' and the urgency cut IPC $P'_{\not\rightarrow_A}$ of P' are depicted in figure 6.4(e) and (f).

We can see that whilst $M_{\not\rightarrow_A}$ and $M'_{\not\rightarrow_A}$ are Markovian branching equivalent, $P_{\not\rightarrow_A}$ and $P'_{\not\rightarrow_A}$ are not...

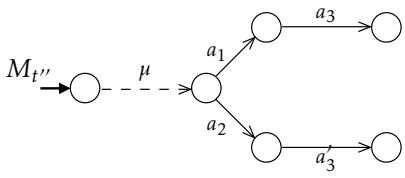
As the example shows, in closed systems, when using constraint-oriented insertion of delays (with the parallel composition), the branching equivalence modulo urgency cut may be always preserved for IMCs, but may be not for IPCs. One may try to provide an explanation to this difference. In both cases, constraint-oriented insertion of a delay in functional models may lead to unexpected timed models, containing nondeterminism. However, this nondeterminism disappears by parallel composition and urgency cut for IMCs, whilst it may not for IPCs.

This difference is mainly due to the way time progresses in IPCs and IMCs. In IPCs, time progresses synchronously, because probabilistic transitions are taken synchronously. Consequently, two subcomponents may have their delays finishing exactly at the same time. Conversely, Markovian transitions are taken asynchronously in IMCs, and the probability that two delays elapse at the same time is zero.

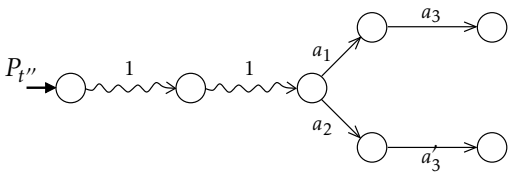
In both IPC and IMC cases, the process in which a delay is inserted in a constraint-oriented way, is not "immediately" aware of the end of the delay. The process is informed of the end of the delay by synchronizing on the end action denoting the end of the delay. In a timed view, the delay elapses, and the synchronization on the end action is taken instantaneously. But the interleaving of interactive transitions in zero-time allows to have actions interleaved before the synchronization on the end action. In this case, those actions may imply non-determinism if their ability to be taken depends whether the delay has elapsed or not.

Inserting Delays Using the Parallel Composition and Hybrid Transitions

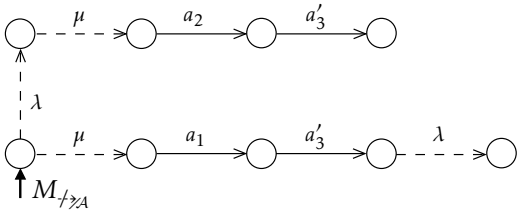
We illustrated that the constraint-oriented insertion of delays with the parallel composition operator of IPCs may lead to wrong results. To avoid this problem, we suggest a new method



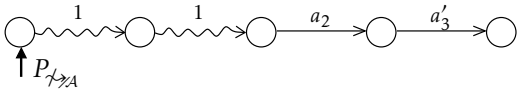
(a) IMC model of the subcomponent C_2



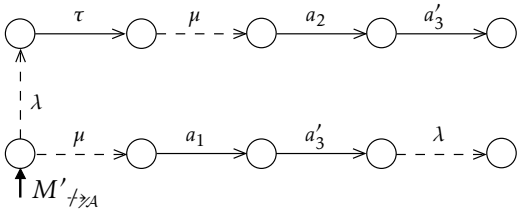
(b) IPC model of the subcomponent C_2



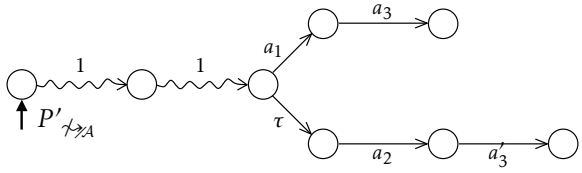
(c) urgency cut IMC of the expected system S



(d) urgency cut IPC of the expected system S



(e) urgency cut IMC of the system S using constraint-oriented insertion of delays



(f) urgency cut IPC of the system S using constraint-oriented insertion of delays

Figure 6.4: Insertion of delays in unclosed IMC and in IPC models using parallel composition

allowing to insert delays by composition, using a composition operator (the hybrid composition operator) different from the parallel composition operator, and that uses a special kind of transition that is both probabilistic and interactive. We call this kind of transition an *hybrid* transition. Because hybrid transitions are only used in replacement of interactive transitions labeled with end action in the delay process:

- In the delay process an hybrid transition merges the end action with the last stage of the discrete phase type distribution of the delay. In other words, the hybrid transition represents both the last probabilistic time step of the delay and the end action.
- in the functional process, an hybrid transition replaces the interactive transition labeled with the end action. For this hybrid transition, the probability is left unspecified.

The two hybrid transitions are then synchronized (on the end action). This synchronization results in a probabilistic transition for which the probability is given by the probability of the hybrid transition of the delay.

Although a model presenting hybrid transitions is, strictly speaking, no longer an IPC, the hybrid composition used for delay insertion removes all hybrid transitions.

This solution can be seen as an adaptation to IPCs of the definition of *active* and *passive* transitions in stochastic process algebras [Hil96]: the hybrid transition in the delay, which has its probability specified, is an active transition, because it forces the probability of the resulting probabilistic transition after synchronization. The hybrid transition in the functional process, which has its probability unspecified, can be seen as a passive transition, because it has an unspecified probability that is forced during synchronization.

This kind of solution generally requires that each synchronization involves at most one active transition. In our case, this requirement is fulfilled: the synchronization involves only two processes (the process and its delay), the process providing a passive hybrid transition and the delay the corresponding active hybrid transition.

Actually, for functional models that should include hybrid transitions with unspecified probabilities, we do not introduce a special transition but let the interactive end transition unchanged. Our functional models need consequently no modification and can be used to get IPC models in a constraint-oriented way. In the following we do not differentiate passive hybrid transitions from interactive transitions.

Example 6.4. *Consider the same system as in example 6.4. The system may take an action a_1 if and only if a delay Δ is not yet elapsed, and an action a_2 may be taken if and only if the delay Δ has elapsed. The delay Δ is modeled by a constant distribution of two time steps.*

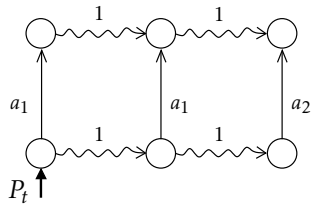
The expected IPC model P_t and the functional model P_f used to process delay insertion are redrawn in figures 6.5(a) and 6.5(b)

The delay $P_{d'}$ to be inserted in a constraint-oriented way is depicted in figure 6.5(c), where a_b and a_e denote the beginning and end actions of the delay, used for composition. Notice that the transition labeled with a_e in the delay is an hybrid one (it is also a probabilistic transition with probability one).

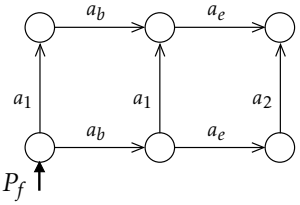
The timed behavior obtained by composition on hybrid transitions is the IPC $P_{\tilde{t}}$ depicted in figure 6.5(d) and equal to $P_f \overline{|[a_b, a_e]|} P_{d'}$, where $\overline{|[]|}$ denote the hybrid composition.

After hiding actions a_b (a_e disappears during the synchronization of the hybrid transitions), the resulting IPC hide (a_b) *in $P_{\tilde{t}}$ is branching equivalent to the expected IPC P_t.*

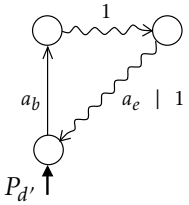
We developed a tool, IPC_INSERT, implementing the composition on hybrid transitions. IPC_INSERT takes as input a process and the delay to insert, both encoded in the BCG format. In the storage



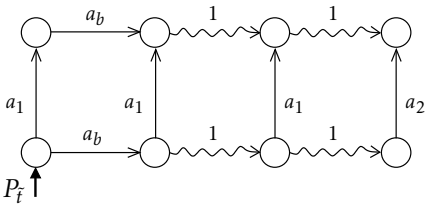
(a) Expected timed behavior



(b) Functional behavior



(c) Delay model



(d) Timed model: $P_f \overline{|[a_b, a_e]|} P_{d'}$

Figure 6.5: Constraint-oriented insertion of delays using hybrid transitions

of the different transitions of an IPC, we added a convention to deal with hybrid transitions:

- a passive hybrid transition is stored as an interactive transition (only the action is stored).
- an active hybrid transition has a label of the form "ACT; prob p", where ACT is the end action to synchronize on, and p the associated probability, p being a real value greater than zero and less or equal to 1.

In addition, IPC_INSERT takes as input the names of begin and end actions used for composing the delay and the process.

IPC_INSERT returns an IPC encoded in the BCG format and corresponding to the process with its delay inserted. In the case there are not other delays to insert, the resulting LTS is an IPC with no more hybrid transitions.

We do not detail algorithmic aspects of IPC_INSERT, this tool being similar to the one implementing the parallel composition of IPCs and presented in the next section.

6.1.5 Parallel Composition of Interactive Probabilistic Chains

We developed a tool, called IPC_COMPOSE, which allows to process the parallel composition of IPCs according to the rules of chapter 5. IPC_COMPOSE is thus similar to the EXP.OPEN tool of CADP. However, EXP.OPEN cannot be used directly to compose IPCs, because the implemented semantics of the parallel composition is different from the semantics of the IPC parallel composition.

This tool takes in input an IPC behavior corresponding to the parallel composition with synchronization of several IPCs. This behavior has to be entered either on the standard input (stdin) or is provided in a text file using the –rules option [1]. The behavior in input is described

1. The possibility to read from stdin the behavior corresponding to the composition to process is justified by the possibility to call IPC_COMPOSE in scripts. It allows to redirect a behavior directly as input of IPC_COMPOSE without creating a (temporary) file

Table 6.1: Example input file for IPC_COMPOSE

```
(
   "PRODUCER_1.bcg"
   |[PUSH_RQ_1,PUSH_RSP_1]|
   "FIFO_1.bcg"
   |[POP_RQ_1,POP_RSP_1]|
   "CONSUMER_1.bcg"
)
|[ ]|
(
   "PRODUCER_2.bcg"
   |[PUSH_RQ_2,PUSH_RSP_2]|
   "FIFO_2.bcg"
   |[POP_RQ_2,POP_RSP_2]|
   "CONSUMER_2.bcg"
)
```

by the grammar, which is a subset of the .exp language:

$$B \ ::= \ (B_0) \ \mid \ B_1 \,|[A]|\, B_2 \ \mid \ B_3$$

where A is a set of actions (not including τ) on which synchronizing. Behaviors B_i must be filenames (between double quotes) of BCG files storing IPCs. The parentheses allow to explicitly express priorities between different synchronizations.

Example 6.5. *An example of an input file for* IPC_COMPOSE*, to generate the graph of two FIFO queues in parallel is given in table 6.1.5. Each queue is connected to a producer and a consumer.* PRODUCER_1.bcg, FIFO_1.bcg, CONSUMER_1.bcg, PRODUCER_2.bcg, FIFO_2.bcg, CONSUMER_2.bcg *are IPCs encoded in the* BCG *format. The actions used for synchronizations are actions of the interactive transitions of the IPCs.*

The output of IPC_COMPOSE is the IPC corresponding to the behavior in input. Its state space is a subset of the Cartesian product of the state spaces of IPCs in input. Suppose that the behavior in input involves n IPCs $\{P_i\}_{0<i\leq n}$, such that $P_i = \langle S_i, \mathcal{A}_i, \rightsquigarrow, \longrightarrow, s_0^i \rangle$. Then, the output IPC $P = \langle S, \mathcal{A}, \rightsquigarrow, \longrightarrow, s_0 \rangle$ is such that $S \subseteq S_1 \times \cdots \times S_n$. In other words, a state of the output IPC corresponds to a combination of states of the input IPCs. In particular, the initial state s_0 of the output IPC is the combination of initial states of the input IPCs, $s_0 = (s_1, \ldots, s_n)$.

In the output IPC P, for every pair of states c and c', we define a valid transition as a tuple $\langle c, l, c' \rangle$ such that:

- either $l \in]0,1]$ or $l \in \bigcup_{i=1}^{n} \mathcal{A}_i$
- $\langle c, l, c' \rangle$ is allowed by the behavior in input, respecting IPC semantic rules of the parallel composition.

We illustrate this notion of valid transition by the following example.

Example 6.6. *Let* $P = \langle S, \mathcal{A}, \rightsquigarrow, \longrightarrow, s_0 \rangle$, $P' = \langle S', \mathcal{A}', \rightsquigarrow, \longrightarrow, s_0' \rangle$ *and* $P'' = \langle S'', \mathcal{A}'', \rightsquigarrow, \longrightarrow, s_0'' \rangle$ *be three IPCs such that* $P = P' \,|[a]|\, P''$*. The initial state* s *of* P *is* $s_0 = \left(s_0', s_0''\right)$*. Suppose that:*

Table 6.2: Algorithm for the parallel composition of IPCs

Input: An IPC behavior involving n IPCs $\{P_i\}_{0<i\leq n}$ such that $P_i = \langle S_i, \mathcal{A}_i, \rightsquigarrow, \longrightarrow, s_0^i \rangle$.

Output: One IPC $P = \langle S, \mathcal{A}, \rightsquigarrow, \longrightarrow, s_0 \rangle$ corresponding to the behavior in input.

Algorithm:

$\mathcal{S}_{\text{exp}} := \emptyset$
$\mathcal{S}_{\text{to\_exp}} := \{s_0\}$
while $\mathcal{S}_{\text{to\_exp}} \neq \emptyset$ **do**
  choose $c = (s_1, \ldots, s_n) \in \mathcal{S}_{\text{to\_exp}}$
  $\mathcal{S}_{\text{exp}} := \mathcal{S}_{\text{exp}} \cup \{c\}$
  $\mathcal{S}_{\text{to\_exp}} := \mathcal{S}_{\text{to\_exp}} \backslash c$
  $\mathcal{S}_{\text{trans}} := \left\{ (c, l, c') \mid c \overset{l}{\rightsquigarrow} c' \text{ or } c \overset{l}{\longrightarrow} c' \text{ is valid} \right\}$
  while $\mathcal{S}_{\text{trans}} \neq \emptyset$ **do**
    choose $(c, l, c') \in \mathcal{S}_{\text{trans}}$
    $\mathcal{S}_{\text{trans}} := \mathcal{S}_{\text{trans}} \backslash (c, l, c')$
    add transition (c, l, c') **in** P
    if $c' \notin \left(\mathcal{S}_{\text{exp}} \cup \mathcal{S}_{\text{to\_exp}}\right)$ **then**
      $\mathcal{S}_{\text{to\_exp}} := \mathcal{S}_{\text{to\_exp}} \cup \{c'\}$
    fi
  od
od

– *from its initial state, P' can take two interactive transitions $s'_0 \overset{a'}{\longrightarrow} s'_1$ and $s'_0 \overset{a}{\longrightarrow} s'_2$, and two probabilistic transition $s'_0 \overset{p}{\rightsquigarrow} s'_3$ and $s'_0 \overset{1-p}{\rightsquigarrow} s'_4$*
– *from its initial state, P'' can take one interactive transition $s''_0 \overset{a}{\longrightarrow} s''_1$ and the single probabilistic transition $s''_0 \overset{1}{\rightsquigarrow} s''_0$*

According to IPC semantics for parallel composition, from its initial state $s_0 = \left(s'_0, s''_0\right)$, P can take one of the following transitions:

– $s_0 \overset{a'}{\longrightarrow} s_1$, *where s_1 is the state* $s_1 = \left(s'_1, s''_0\right)$
– $s_0 \overset{a}{\longrightarrow} s_2$, *where s_2 is the state* $s_2 = \left(s'_2, s''_1\right)$
– $s_0 \overset{p}{\rightsquigarrow} s_3$, *where s_3 is the state* $s_3 = \left(s'_3, s''_0\right)$
– $s_0 \overset{1-p}{\rightsquigarrow} s_4$, *where s_4 is the state* $s_4 = \left(s'_4, s''_0\right)$

A state s is in the state space S of the output IPC P, if, from the initial state $s_0 \in S$, there is a sequence of valid transitions allowing to reach it.

IPC_COMPOSE constructs the output IPC by iteratively exploring all sequences of valid transitions from the initial state s, which corresponds to a classical approach for constructing graphs. The main lines of the algorithm used in IPC_COMPOSE are presented in table 6.1.5.

In addition to the generation of the parallel composition of IPCs, we integrated the following options in IPC_COMPOSE:

- option -m allows graphical monitoring of the generation
- option -p applies on-the-fly maximal progress cut to the generated IPC, and should be used only when we are targeting an IPC probabilistic branching bisimilar to the one obtained without this option
- option -u applies on-the-fly urgency cut to the generated IPC, and should by used only if the generated IPC is closed

6.1.6 Compositional Interactive Probabilistic Chain Modeling

In practice, we are modeling systems following a compositional approach. The idea is to use the bisimulations to reduce the intermediate state space of compositions. Efficient minimization algorithms, with respect to bisimulations, exist [GH02, HS99, BdS92, GV90]. The reduction according to probabilistic strong bisimulation is already implemented in the tool BCG_MIN of the CADP toolbox, with option -prob. Adding the -branching option to BCG_MIN allows a reduction according to the probabilistic branching bisimulation.

The compositional approach for generating IPCs is summarized in figure 6.6. Functional models stored in the BCG format are enriched with delays modeled as phase-type distribution and also stored in the BCG format. Functional models and delay models may be obtained by compilation of LOTOS specifications using CAESAR.ADT and CAESAR compilers. The insertion of delays is processed in a constraint-oriented way, using IPC_INSERT, to avoid to introduce nondeterminism. Then, the system studied is generated by alternating phases of composition of subcomponents using IPC_COMPOSE, and of minimization using BCG_MIN with option -prob. When we want to preserve only the branching minimization along all compositions, maximal progress cut option -p can be used with IPC_COMPOSE, and the option -branching can be used with BCG_MIN. Using these options may reduce the generated IPCs significantly.

6.2 Computation of the Latency Distribution

In chapter 5 we presented a transformation allowing to associate a DTAMC to an IPC. As illustrated in chapter 2, a DTAMC can be used to compute performance measures, such as a latency distribution (cf. chapter 3).

In an IPC, the latency is defined as the time elapsed between two actions (called begin action and end action), i.e., between two interactive transitions. We developed a tool, called IPC_DISTRIBUTION, allowing to return the distribution of a latency in an IPC. IPC_DISTRIBUTION implements the computation of the distribution of the latency corresponding to the time between an action a_b (begin action) and the first occurrence of an action a_e (end action). It corresponds to the definition of the latency as given in [CHLS09] (i.e., it is a particular case of the generalization given in chapter 3); hence it does not allow to compute end-to-end latencies. According to chapter 3, the basic software architecture (shown in figure 6.7) of the IPC_DISTRIBUTION tool is structured into five separated steps:

(1) Transformation of the input IPC into a DTAMC
(2) Computation of the normalized steady state probabilities
(3) Extraction of absorbing Markov chains between states allowing to take the begin action and states allowing to take the end action
(4) Computation of the distribution of the time before absorption in extracted Markov

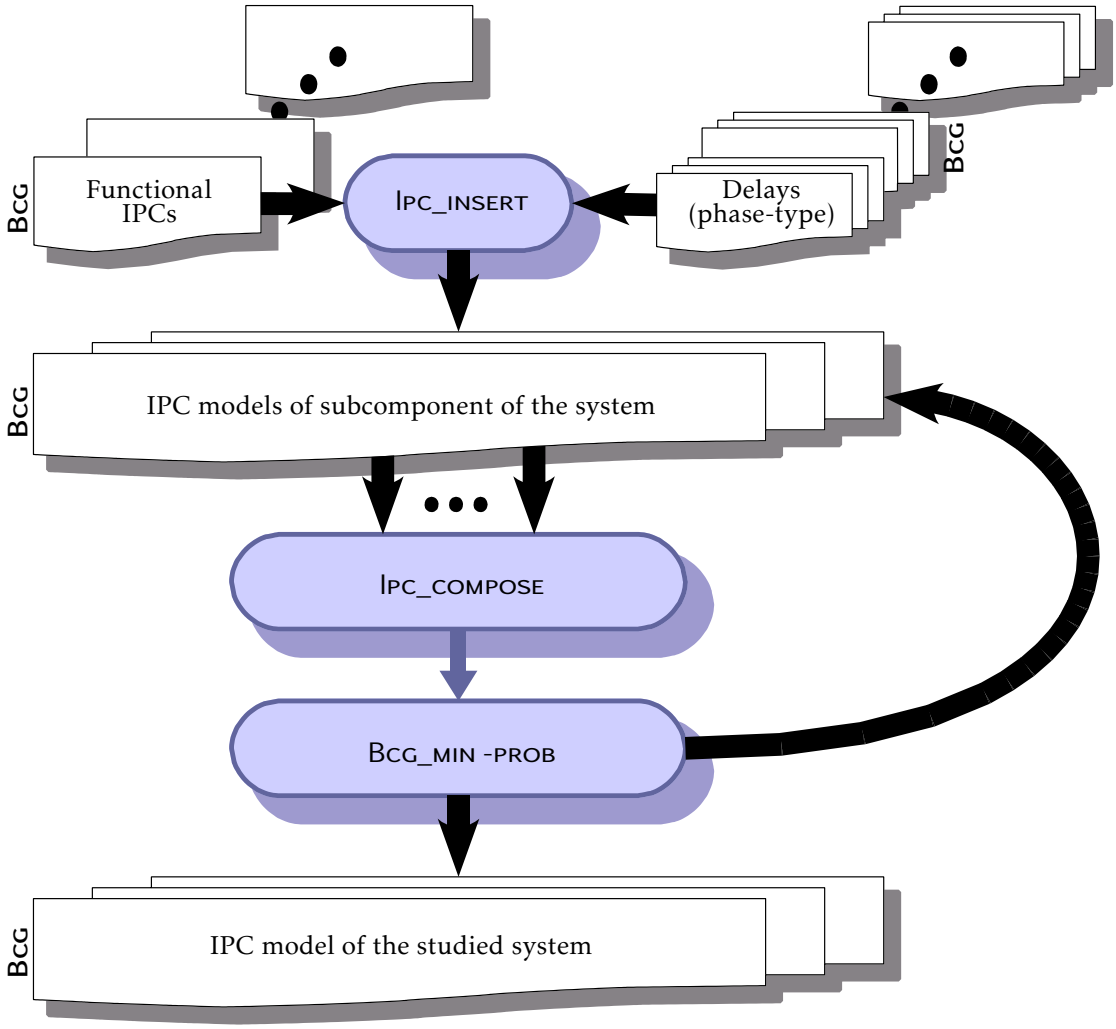


Figure 6.6: IPC modeling flow

chains

(5) Computation of the latency distribution as sum of computed distributions weighted by normalized probabilities

Steps (2) to (5) are those presented in chapter 3 to compute a latency distribution in a DTAMC.

The BCG format of the CADP toolbox is efficient and well-adapted to store arbitrary directed graphs (for instance LTS, IMCs, or IPCs), with annotations on transitions. Unfortunately, the BCG format does not allow the association of information to states as would be required for state annotations in a DTAMC, on which our performance methodology is based on. To avoid to define a new graph format for DTAMCs, we use the transformation of DTAMCs into SMCs as presented in section 5.2.6. Because there is no information associated to states in a SMC, it can be stored in the BCG format. To store a SMC in the BCG format, we use the convention "ACT; prob p [C]" for the string associated to a labeled probabilistic transition, where ACT is the action associated to the transition, p its probability, and C the value of the constant distribution associated to the transition (if the distribution is $\mathbf{C}_n$, we have C = n).

For step (2), we recall that we are able to get the same results concerning steady state probabilities, with the associated DTAMC or its associated SMC (cf. lemma 5.9). Our implementation merges steps (3) to (5) to directly obtain an absorbing DTMC $\mathcal{M}$ representing the targeted latency distribution (as a phase-type distribution). To this aim, the SMC is transformed as following:

- Two additional states are added: a state s_0, corresponding to the initial state of $\mathcal{M}$, and a state s_a corresponding to the absorbing state of $\mathcal{M}$
- We identify the set α of states allowing to take the begin action, which correspond to initial states of absorbing chains that would be extracted
- We also identify the set ω of states allowing to take the end action, which correspond to absorbing states of the chains that would be extracted
- For each state $s \in \alpha$, we add a transition $s_0 \overset{p}{\rightsquigarrow} s$, with p the normalized steady state probability associated to s
- For each state $s \in \omega$, each transition $s' \overset{p}{\rightsquigarrow} s$ (for some s' and some $p > 0$) is replaced by a transition $s' \overset{p}{\rightsquigarrow} s_a$
- States unreachable from s_0 (i.e., states that are not in a path between a state of α and a state of ω) are removed.
- Finally, the graph is transformed into an absorbing DTMC by removing interactive transitions (for instance by hiding them and using minimization according to the branching bisimulation)

The absorbing DTMC $\mathcal{M}$ is a discrete phase-type distribution corresponding to the targeted latency distribution. The distribution of the latency, i.e., the distribution of the time before absorption in $\mathcal{M}$ is finally computed by transient analysis of the absorbing state s_a. The distribution of the latency is the distribution computed on $\mathcal{M}$ (shifted by one because we added an additional transition from s_0 to each state of α). The implemented software architecture of the IPC_DISTRIBUTION tool is depicted in figure 6.8.

In our implementation, IPC_DISTRIBUTION takes as input an urgency cut deterministic IPC, and the names of begin and end actions identifying the latency. As output, IPC_DISTRIBUTION returns the latency distribution in a text format, easily understandable by graph plotting tools such as GNUPLOT. The range of the distribution computation is given by the option -n (by default the distribution is computed between 0 and 30). Moreover, the minimum, average and

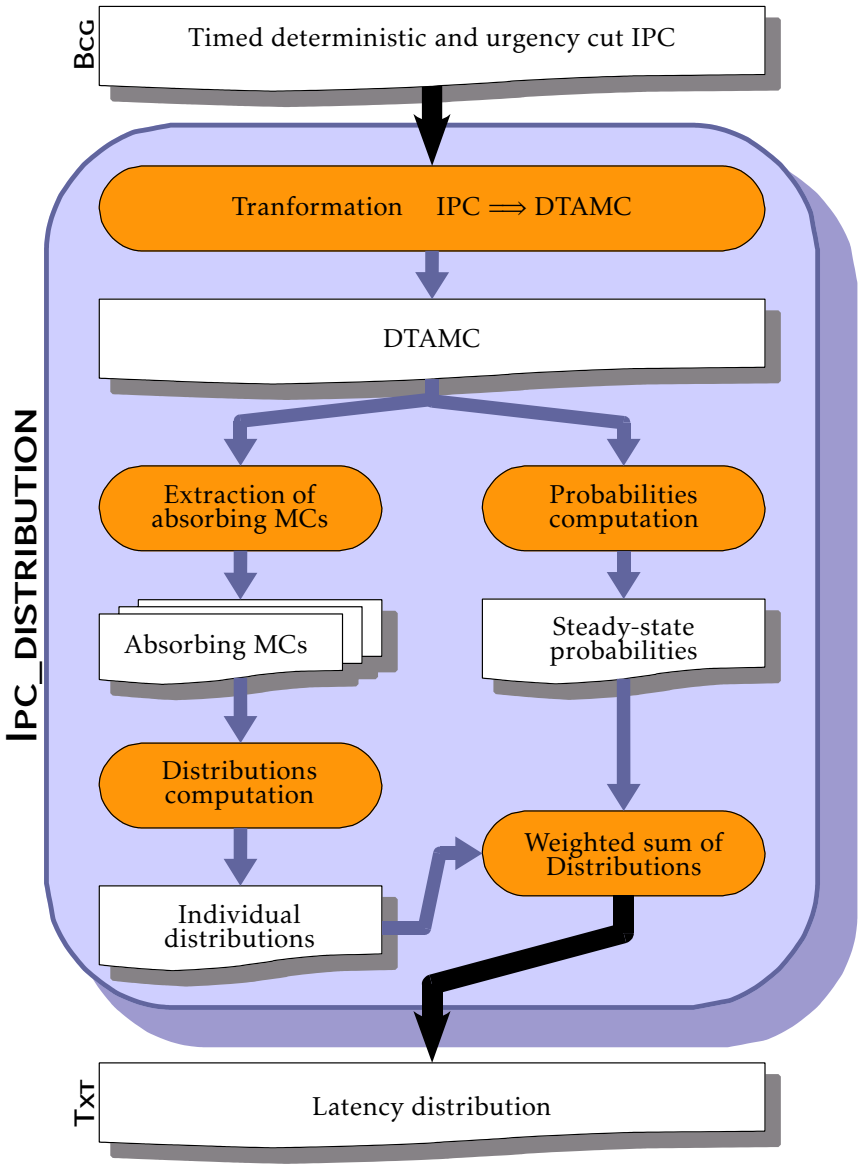


Figure 6.7: Basic architecture the IPC_DISTRIBUTION tool

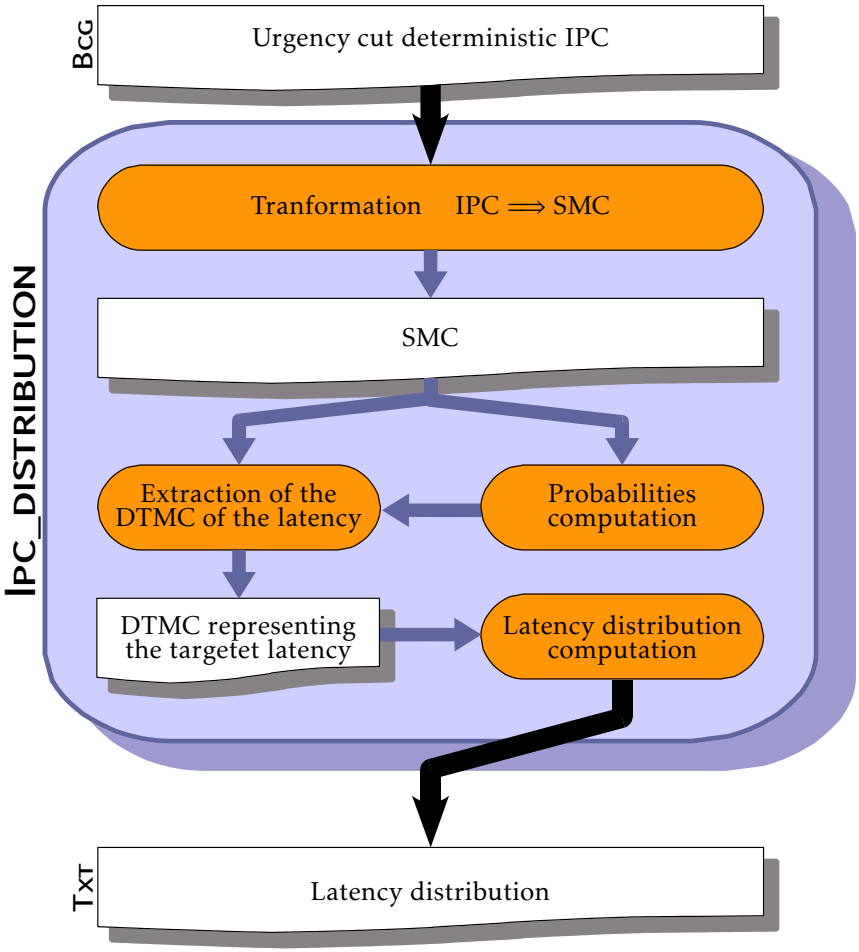


Figure 6.8: Architecture of the implemented IPC_DISTRIBUTION tool

maximal values of the distribution are returned. The minimum value is the simplest measure to compute: it corresponds to the shortest path to reach an absorbing state in the absorbing DTMC. Conversely, the maximum value corresponds to the longest path and may be infinite if there are loops of probabilistic transitions. The returned average value is only an approximation since it is the average of the values of the computed range of the distribution.

6.3 Discussion

In this chapter, we presented the toolchain supporting the IPC formalism in practice. In a first section, we presented how we generate IPC models by reusing existing tools of the CADP toolbox. In a second section, we presented the tool IPC_DISTRIBUTION that implements the computation of the distribution of a latency, as defined in chapter 3.

To get IPC models, we authorize to use LOTOS processes instead of IPC processes for the part of the LOTOS semantics that is shared by the IPC semantics. By this way, we can reuse CAESAR and CAESAR.ADT compilers of the CADP toolbox to generate IPCs stored in the BCG format. Unfortunately, some IPC operators cannot be taken into account by writing LOTOS processes. In particular, the nondeterministic choice operator can only be used partially (we cannot write a nondeterministic choice between several probabilistic choices), and the parallel composition operator is forbidden. We implemented the parallel composition in a separated tool, IPC_COMPOSE, which directly proceeds the parallel composition of IPC models (stored in the BCG format).

We also underlined a problem in the modeling phase, encountered when inserting time in functional models, following a constraint-oriented methodology, i.e., using the parallel composition of IPCs. This problem is inherent to the constraint-oriented insertion of delays: a process in which a delay is inserted is not aware of the end of the delay after the last time step of the delay, but after synchronizing on the end action of the delay. Consequently, due to the interleaving semantics of parallel composition, some interactive transitions may be interleaved before taking the end action. Results may be rendered false (nondeterminism is introduced) when those interleaved interactive transitions depend on the end of the delay. We provided a solution to this problem by inserting delays in a constraint-oriented way but using hybrid transitions, i.e., both probabilistic and interactive transitions, which are used to denote the last time step of a delay and the end action on which the process synchronizes with its delay. This solution is similar to the active and passive transitions mechanism, introduced in [Hil96]. We implemented it in the tool IPC_INSERT. Another solution to this problem could be the use of a scheduler. Indeed, we could force the end action to be taken atomically with the last time step of the delay, i.e., no other interactive transition is allowed to be interleaved between the last time step of a delay and the end action on which the process synchronizes.

Chapter 7

Industrial Case-Study: The xSTream Architecture

In this chapter, we apply the IPC formalism to investigate the xSTream architecture. The xSTream architecture, designed at STMicroelectronics, is a multiprocessor data-flow architecture for high performance embedded multimedia streaming applications. The study of the xSTream architecture takes place in the context of the Multival project [CGH+08], whose aim is to apply formal methods for functional verification and performance evaluation of multiprocessor multithreaded architectures.

In a first section, we present the xSTream architecture. We detail its targeted programming model and its implication on design choices. In a second section, we present two functional models of the architecture, at different levels of abstraction, detailing the modeling choices to circumvent the state space explosion problem. In a third section, we present the enrichment of those functional models to allow performance evaluation, and we use the performance flow presented in chapter 6 to study some latencies in those models.

7.1 Presentation of the xSTream Architecture

For many multimedia applications, efficiency and programmability are required. Those two requirements are opposed in the sense that, in general, the quest of efficiency worsens programmability. Parallelism is mandatory to increase efficiency, in particular for systems presenting strong limitations concerning power consumption (and thus frequency). Unfortunately, parallelism generally leads to more difficulties to exploit resources in the most efficient way. However, for the particular case of data-flow applications, the stream-oriented programming model is known to be well-adapted to parallel architectures, because parallelism and locality of data are explicitly exposed by the programmer.

7.1.1 Stream-Oriented Programming Model

The stream-oriented programming model is closely related to data-flow programming, and consists in performing several computation steps, called *filters* on a stream of data. The set of all filters can be seen as a network of pipelines through which the stream of data is processed. Filters communicate through (unbounded) FIFO queues: a filter reads data from one or several

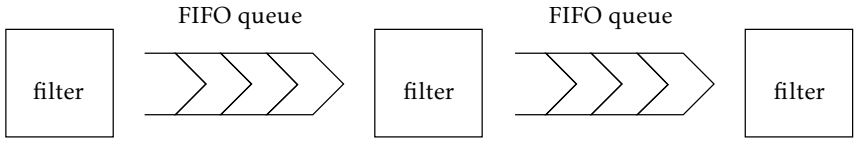


Figure 7.1: The stream-oriented programming model

input queues, processes it, and write its result into one or several output queues. There is consequently no need to manage data coherency: a filter is only authorized to read and write data in its input and output queues. During its processing phase, the filter may use local variables. In addition, there is no synchronization needed for filters to communicate with each others, since they are implicitly realized by the FIFO queues.

By dividing an application into filters, one can exploit parallel architectures more easily. An example of processing of a data-flow using the stream-oriented programming model is depicted in figure 7.1.

An efficient execution of a stream-oriented application requires:
– an optimized slicing of the application into filters ensuring maximal parallelism
– an efficient management of the FIFO queues used for the communication between filters

7.1.2 Design of the xSTream architecture

The xSTream architecture provides an adequate support for stream-oriented programming. It consists of a bunch of multi-threaded *processing elements* (xPE) (i.e., processors with some local memory (LM)), communicating over a *network-on-chip* or *NoC* (xSTNoc). The parallelism proposed by the set of processing elements can be exploited by the stream-oriented programming model: filters are mapped on different processing elements. In addition, according to the workload of each processing element, several filters can be mapped on the same processing element. The xSTream architecture also provides a hardware support for the communications in the stream-oriented programming language, i.e., for the FIFO queues used to communicate between filters. Each processing element is linked to the NoC through buffering hardware queues for input streams (called *pop queues*) and for output streams (called *push queues*). Push and pop queues of a processing element are grouped inside a *flow controller* (xFC) that manages their access to the NoC. The xSTream architecture is depicted in figure 7.2.

Between two processing elements on which communicating filters are mapped, the tuple (push queue, path in the NoC, pop queue) is supposed to behave like a FIFO queue. We call this tuple a *virtual queue*.

Because it is obviously not possible to provide infinite FIFO queues between filters mapped over processing elements, the size of a virtual queue is physically limited by the size of the push queue plus the size of the pop queue plus the length of the path in the NoC. Nevertheless, push and pop queues can be extended into the *local memory* (LM) of their respective flow controllers, to provide a larger virtual queue. Thus, a queue stores its elements either in its dedicated hardware or in the local memory — the latter being much less efficient. This extension, called *backlog mechanism*, provide a trade-off between the size of the virtual queue and its cost in terms of hardware implementation. The transfer of elements from backlog memory to the hardware queue is automatically managed by the flow controller. Although the backlog mechanism authorizes larger queues, they remain bounded size. To avoid their overflow, in-

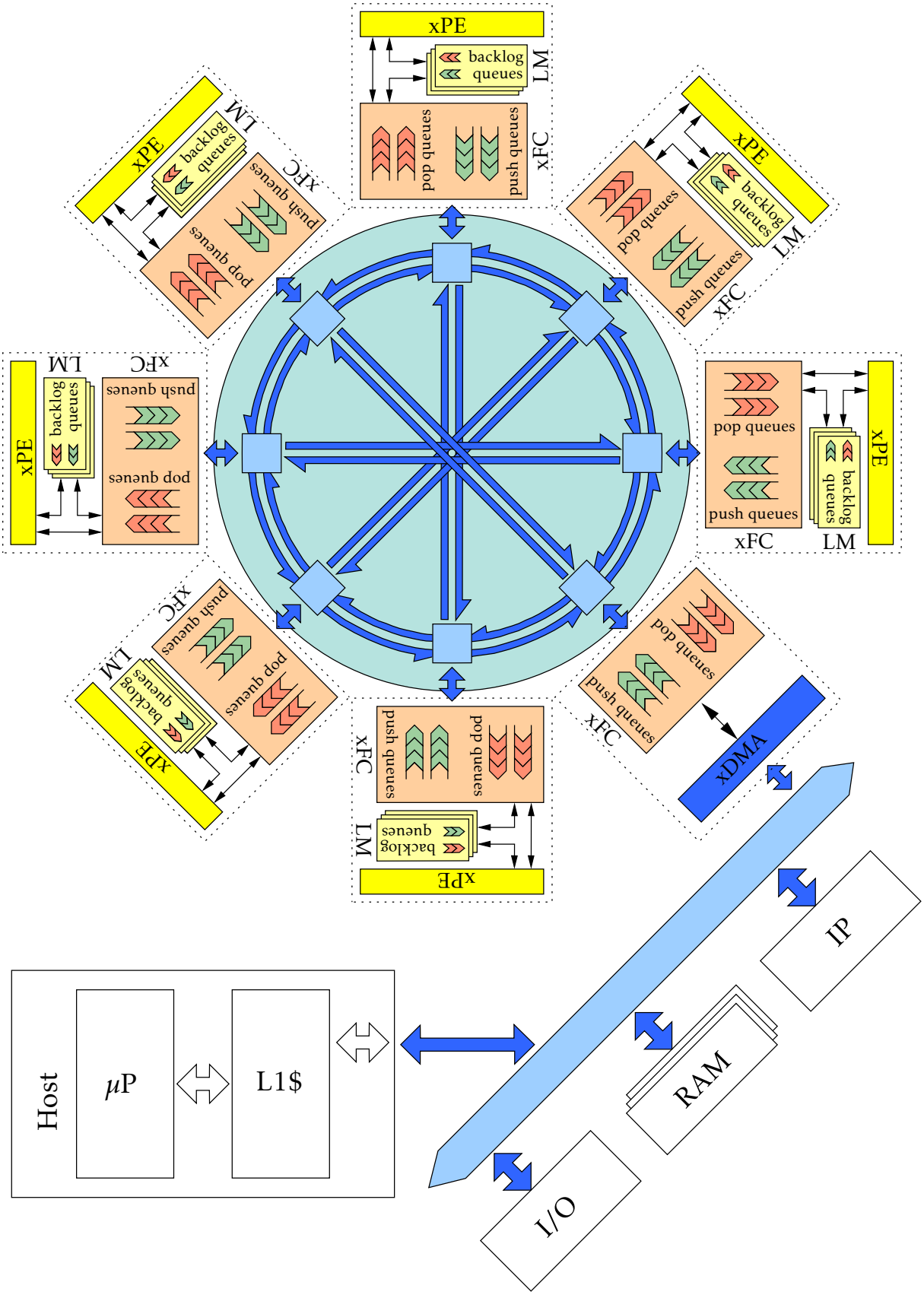


Figure 7.2: The xSTream architecture

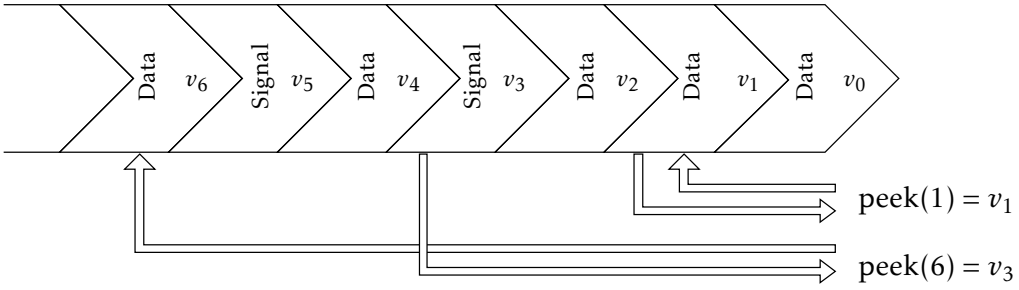


Figure 7.3: Illustration of the behavior of the peek operation in a pop queue where $\{v_i\}_{i\in\mathbb{N}}$ are values (which are either data or signals) carried by the queue

sertion operations on xSTream queues are blocking. A *push operation* (insertion of an element) on the queue is blocked until there is a free place. Similarly, a *pop operation* (removal of an element) on a pop queue is blocked until an element is available.

The push and pop queues and the NoC are the central point of the communication infrastructure of the xSTream architecture. On their efficiency relies the performance of the system.

7.1.3 Pop queues

Pop queues of the xSTream architecture are not simple FIFO queues as we could have supposed according to the stream-oriented programming model. Indeed, in many streaming applications, data is not necessarily accessed in the order they are stored in the input FIFO queue. If a filter wants to access an element that is not at the head of its pop queue, one solution could be to pop all the elements from the queue until the desired one is accessed, and, during this time, to store the others in local memory. For efficiency reasons (to avoid to have to remove elements from the queue and to store them in the local memory), a pop queue bypasses the FIFO mechanism, and provides the *peek operation*, allowing to read an element at any place of the pop queue. Contrary to the pop operation, the element returned by a peek operation is only read, and not removed from the queue.

In a streaming application, it is usual to distinguish data significant information, processed by the application, from control information used by the application algorithm. For instance, control information can be used to differentiate different frames in a stream. The xSTream architecture supports the distinction between these two kinds of information. Control information is called *signal*, contrary to data information, simply called *data*. This distinction impacts also the behavior of a peek operation.

The peek operation allows to access an arbitrary element in the pop queue. Consider the operation "peek(n-1)" to read the n-th element of the pop queue (elements are numbered from 0). If all elements before the n-th place in the queue are data, the peek operation returns the n-th element. If there are signal elements available before the n-th place in the pop queue, the peek operation returns the first available signal element before the n-th place of the pop queue. The behavior of the peek operation is depicted in figure 7.3.

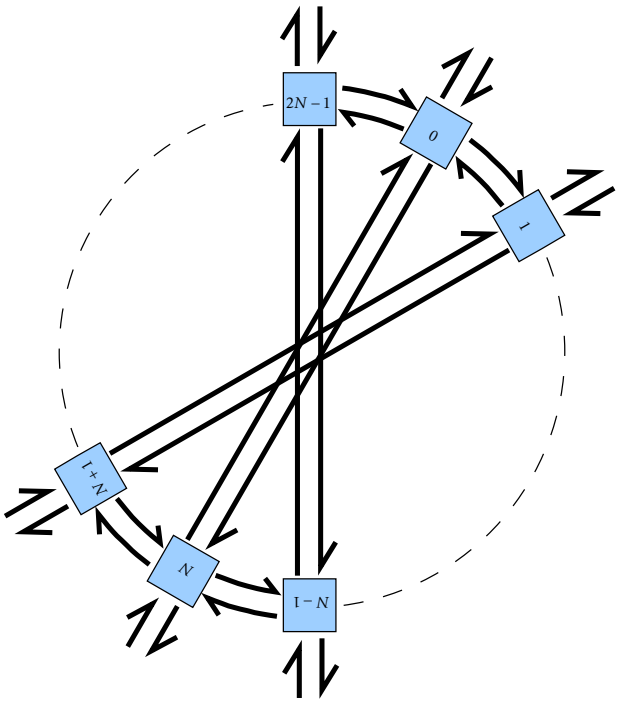


Figure 7.4: The spidergon topology of the xSTream NoC

7.1.4 Network-on-Chip

The NoC of the xSTream architecture is a set of routers interconnected in the aim of routing data-flows between the different processing elements (actually between the different flow controllers). The topology of the xSTream NoC is a spidergon [CLM+04]: all routers are disposed along a ring, each router presenting four communication ports: one port connected to a processing element, and three ports for the communication with other routers. In this spidergon topology, a router can communicate over a point-to-point link:

- with the preceding router on the ring over its *left link*
- with its succeeding router on the ring over its *right link*
- and with the router that is diametrically opposed over its *across link*

The spidergon topology of the xSTream NoC limits the number of communication ports of each router, but also limits the number of communication links needed to pass from a router to an other one. The spidergon topology is depicted in figure 7.4.

In the xSTream NoC, different flows may be multiplexed on the same hardware link. In addition, the routing algorithm of the xSTream NoC is a static source routing. Through the NoC, the path between two processing elements is static but it minimizes the number of links occupied. A path on the NoC is composed of, at most, one across link, which, if present, has to be the first link or the last link used in the path. In addition, to ensure that the NoC is deadlock free [CLM+04], a path in the NoC is not allowed to pass through the node 0. For a given application, between two filters, a single path is followed by all the elements of the flow. On the point of view of an application, the NoC behaves like a FIFO queue: elements follow the same path and go out of the NoC in the order they entered it.

Since processing elements are multithreaded, one can imagine to have several different streams between the same pairs of processing elements, i.e., to have several streams sharing the same path on the NoC. This functionality is enabled in xSTream by the virtual channel concept.

7.1.5 Credit Protocol

A protocol between push and pop queues of the same virtual queue is dedicated to flow control. This protocol, called the *credit protocol*, aims at ensuring that there is no more element sent by the push queue than the pop queue can store. In other words, an element entering the NoC is ensured to be able to go out of the NoC as soon as possible (i.e., the element is not blocked in the NoC waiting for a free place in the pop queue).

The credit protocol relies on counters and thresholds managed by the push and the pop queue, and *credit messages* sent from the pop queue to the push queue. We call the path on the NoC from the push queue to the pop queue the *data path*, and the path in the revert way from the pop queue to the push queue is called the *credit path*.

The behavior of the credit protocol can be briefly given by the separated behaviors of the push queue and of the pop queue. On one side, the push queue manages a counter, the *push counter*, representing the number of available places in the pop queue. Initially, the push counter is set to the size of the pop queue. At any time, the push queue can receive a credit message from the pop queue containing a number of additional elements the push queue is authorized to send. When receiving this message, the push counter is increased by the value received in the credit message. Each time an element goes out of the push queue, the push counter is decremented by one. On the pop queue side, a counter of consumed elements, the *pop counter* is managed. Initially, the pop counter is set to zero, and each time an element is consumed (i.e., withdrawn from the pop queue), the pop counter is incremented by one. The pop queue further depends on a (constant) *credit threshold* that is less or equal to its size. Two situations may lead to the emission of a credit message:

- When the pop counter is greater or equal to the credit threshold, a credit message can be sent by the pop queue. This credit message contains the value of the credit threshold. As soon as the credit message is sent, the pop counter is decremented by the value of the credit threshold.
- If there is a peek operation concerning an element not available in the queue, a credit message can be sent (for instance there are two elements available in the queue and there is a peek operation for the third element of the queue). This credit message contains the value of the pop counter. As soon as the credit message is sent, the pop counter is reseted to zero.

With respect to the credit threshold of the pop queue, one can distinguish two extreme behaviors:

- **credit threshold set to one.** Each time an element is consumed, a credit message is sent from the pop queue to the push queue. This behavior induces that a lot of control messages are sent to the NoC, reducing its performance.
- **credit threshold set to the size of the pop queue.** This configuration minimizes the number of credit messages sent from the pop queue. Nevertheless, it has an impact on the performance of the virtual queue: each time the push credit runs out, the push queue is blocked until receiving a credit message. This message is send only when all the

elements sent by the push queue are consumed in the pop queue.
According to those remarks concerning the value of the credit threshold, one has to find a trade-off allowing to minimize communication over the NoC, but preserving the performances of the virtual queue.

In the initial specification of the xSTream architecture, the credit protocol was optional. At the cost of an overhead, this protocol enables a flow control and reduces contentions in the NoC (there are no elements of a data-flow blocked in the NoC). However, the functional verification, performed at STMicroelectronics, of the models we present in the next section showed that the credit protocol is actually mandatory to avoid deadlocks provoked by badly written applications. Consequently, all models we study in the following sections, in particular performance models, take the credit protocol into account.

7.2 Functional Models of the xSTream Architecture

In this section, we present the formal model used to study the xSTream architecture in a hierarchical and compositional way. Our aim is to study the interaction between two virtual queues between the same pair of processing elements, i.e., two parallel streams between two pairs of filters mapped on the same processing elements. According to the characteristics of the NoC, the two streams share the same data path and the same credit path on the NoC. We abstract data path and credit path on the NoC by standard FIFO queues. We construct this model by successive refinements of a simple virtual queue, i.e., a push queue directly connected to a pop queue. For each refinement, we compare the result to the model of the simple virtual queue, according to the branching bisimulation for LTS, using Bisimulator of the Cadp toolbox.

7.2.1 Modeling choices

To model large systems such as the xSTream architecture, abstraction is needed. Firstly, we target only the communication architecture. In other words, we do not focus on the behavior of processing elements but only on virtual queues used for communication. Consequently, we are not interested in the data values transported between filters, but in the way data are transported.

Luckily, the behavior of a virtual queue is, in term of possible communication, independent from the value that is carried in the communication itself. The global behavior of a virtual queue consists in carrying the values in the right order (FIFO scheme). However, because the behavior of a pop queue relies on the distinction between data and signal for the peek operation, we cannot consider that all transported elements are equal, but we have to distinguish data from signals. In our models, the elements carried over the communication architecture of xSTream can take two values: data or signal. This first abstraction allows to circumvent the state space explosion problem by limiting the state space of values carried by the architecture.

In a purely functional model of the xSTream architecture, we do not model applications (i.e., filters). For a virtual queue, all possible interactions with its environment (i.e., with filters) may happen. However, we chose not to model the peek operation behavior on the pop queue. This choice has not a great impact on the state-space of the model of a pop queue. Because peek operations do not modify the pop queue, peek operations correspond to loops on the states of

the model. But this choice has an impact on the complexity of the credit protocol: a credit message is only sent when the pop counter exceeds the credit threshold. Nevertheless, we keep the distinction between data and signals.

In our models, all operations follow an handshake scheme: the operation is initiated by a request and when it is processed, a response is sent back. In addition, several requests cannot be pipelined: a new request is sent only if the response of the last operation has been received. Additionally, operations are used for communications, i.e., to exchange information. When the initiating process wants to send information, the information is associated to the request, and the response is an acknowledgment ensuring that the information has been received. When the initiating process wants to receive information, the request is sent to ask for the information, and the information is returned on the response.

To permit different virtual queues sharing the same path on the NoC, we have to model virtual channels of the NoC. We associate a pop queue identifier to each element withdrawn from a push queue. A pop queue only accepts elements presenting its own identifier in input. Similarly, a push queue identifier is associated to each credit message sent by a pop queue, and a push queue only accepts credit messages presenting its own push queue identifier.

7.2.2 Push Queue Model

The push queue model, depicted in figure 7.5, has three interfaces: an input, or push, interface (I, with gates PUSH_RQ and PUSH_RSP), an output interface (O, with gates OUT_RQ and OUT_RSP), and a credit interface (C, with gates CREDIT_RQ and CREDIT_RSP). The LOTOS model of a push queue is given in section E.2 of appendix E. A push queue is characterized by four parameters:

- Size: the queue size, corresponding to the number of elements the push queue can store
- Credit: the initial value of the push counter for the credit protocol (usually corresponding to the size of the associated pop queue)
- $\mathsf{Id}_{\mathsf{push}}$: the identifier of the push queue
- $\mathsf{Id}_{\mathsf{pop}}$: the identifier of the pop queue, associated to the push queue

The three interfaces are used with some offers on the LOTOS gates:

- on I: $\langle E \rangle$. An element E is waited (either a data or a signal).
- on O: $\langle E, \mathsf{Id}_{\mathsf{pop}} \rangle$. An element E is sent with the identifier $\mathsf{Id}_{\mathsf{pop}}$ of the associated pop queue.
- on C: $\langle N, \mathsf{Id}_{\mathsf{push}} \rangle$. A credit message is waited, with N the credit value and $\mathsf{Id}_{\mathsf{push}}$ the identifier of the addressee push queue.

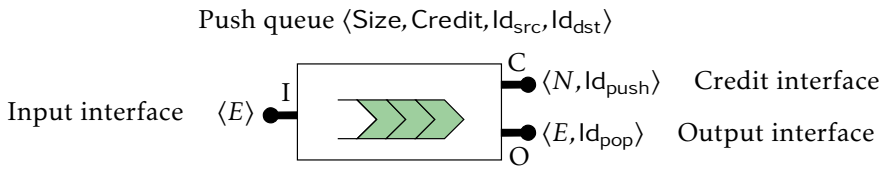


Figure 7.5: The interfaces of a push queue

The behaviors of the different interfaces of a push queue are the following:

- The push interface is used to process a push operation on the queue. The push queue is always able to receive an element (data or signal) from its environment on the PUSH_RQ gate. After synchronizing on PUSH_RQ, if there is a free place in the queue, the element is inserted and a PUSH_RSP is sent back. in the case the push queue is full, the

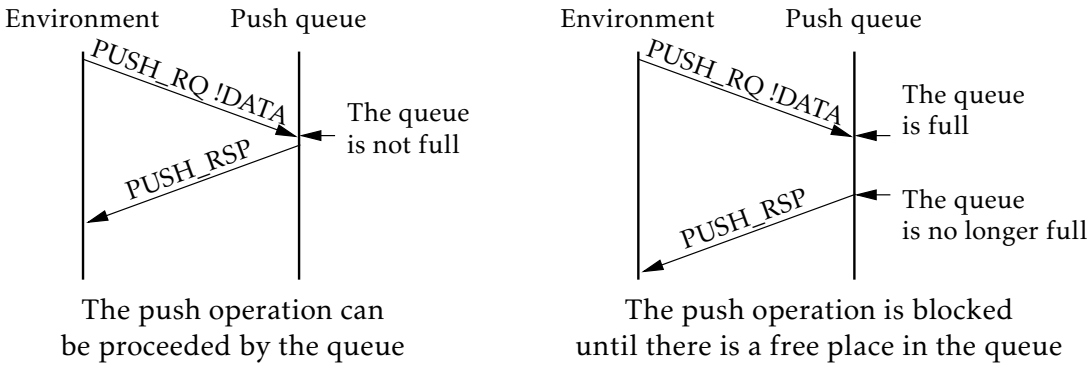


Figure 7.6: Illustration of the blocking behavior of the push operation

queue is blocked, waiting for a free place before inserting the element and returning the PUSH_RSP. The blocking behavior of a push operation is illustrated by the sequence diagrams of figure 7.6.

– The output interface is used to withdraw an element from the queue. The push queue initiates the withdrawal of its elements by providing its first element to its environment on the OUT_RQ gates. In addition, on the same OUT_RQ gate, the push queue also provides the identifier of the pop queue that would receive the element. When the environment accepts the element, it acknowledges on the OUT_RSP gate, which induces the effective removal of the first element in the push queue and the update of its credit counter (it is decreased by one).
– The credit interface is used to receive credit messages from the associated pop queue. The reception is effective on the CREDIT_RQ gate, only if the push queue identifier associated to the credit message is the own identifier of the push queue. In this case, the push queue acknowledges on the CREDIT_RSP gate and updates its push counter.

7.2.3 Pop Queue Model

The pop queue model, depicted in figure 7.7, has three interfaces: an input interface (I, with gates IN_RQ and IN_RSP), an output, or pop, interface (O, with gates POP_RQ and POP_RSP), and a credit interface (C, with gates CREDIT_RQ and CREDIT_RSP). The LOTOS model of a pop queue is given in section E.3 of appendix E. A pop queue is characterized by four parameters:

– Size: the queue size, corresponding to the number of elements the pop queue can store
– Threshold: the credit threshold of the pop queue
– $\mathsf{Id}_{\mathsf{pop}}$: the identifier of the pop queue
– $\mathsf{Id}_{\mathsf{push}}$: the identifier of the push queue, associated to the pop queue

The three interfaces are used with some offers on the LOTOS gates:

– on I: $\langle E, \mathsf{Id}_{\mathsf{pop}} \rangle$. An element E is waited, with the identifier $\mathsf{Id}_{\mathsf{pop}}$ of the addressee pop queue.
– on C: $\langle N, \mathsf{Id}_{\mathsf{push}} \rangle$. A credit message is sent by the pop queue, with N the credit value and $\mathsf{Id}_{\mathsf{push}}$ the identifier of the addressee push queue.
– on O: $\langle E \rangle$. An element E is withdrawn.

The behaviors of the different interfaces of a pop queue are the following:

– The pop interface is used to process a pop operation on the queue, i.e., to withdraw an element from the queue. The pop queue is always able to receive, from its environment,

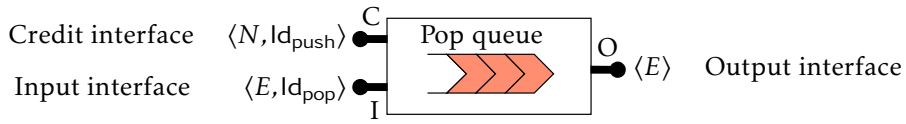


Figure 7.7: The interfaces of a pop queue

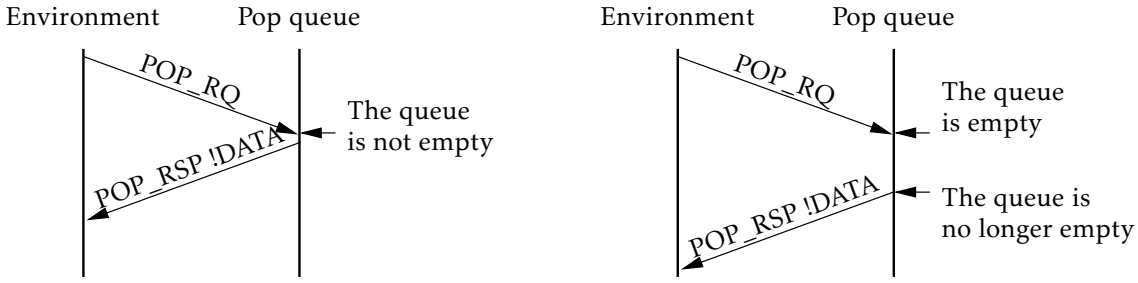


Figure 7.8: Illustration of the blocking behavior of the pop operation

a request to remove an element, POP_RQ. When the queue can grant the pop operation (i.e., there is at least one element available in the queue), it sends the element on the POP_RSP gate, the element is effectively removed from the queue, and the pop counter is updated (it is increased by one). The blocking behavior of a pop operation is illustrated in figure 7.8.

– The input interface is used to insert elements in the pop queue. The pop queue is always able to receive an element on the IN_RQ gate. This synchronization is enabled only if the pop queue identifier associated to the sent element is the identifier of the queue. After synchronizing on IN_RQ, if there is a free place in the queue, the element is inserted and a IN_RSP is sent back. In the case the pop queue is full, the queue waits for a free place before inserting the element and returning the acknowledgment IN_RSP.
– The credit interface is used to sent credit messages to the associated push queue. A credit message is emitted on the CREDIT_RQ gate, associated to the push queue identifier of the queue that would receive the message. Then, the pop queue waits for an acknowledgment CREDIT_RSP from its environment. Its pop counter is updated at the synchronization on CREDIT_RSP.

7.2.4 Simple Virtual Queue

The coarsest abstraction of a virtual queue in the xSTream architecture is obtained by directly connecting a push queue to a pop queue. It corresponds to what we called a *simple virtual queue*. The output interface (O) of the push queue is linked to the input interface (I) of the pop queue, and credit interfaces (C) of the two queues are connected together. A simple virtual queue is depicted in figure 7.9.

The simple virtual queue behaves almost like a FIFO queue, but its behavior depends on the credit threshold. For a credit threshold set to one, when hiding all internal transitions (credit interfaces of push and pop queues and push queue output/pop queue input interfaces), the simple virtual queue is branching equivalent to a FIFO queue. For instance, consider a

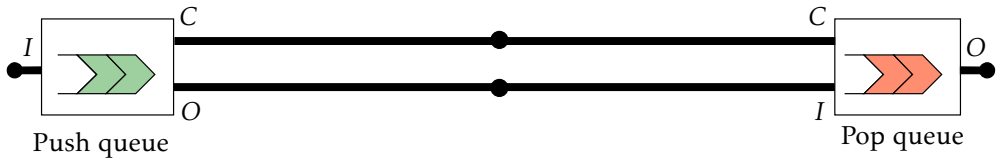


Figure 7.9: A simple virtual queue

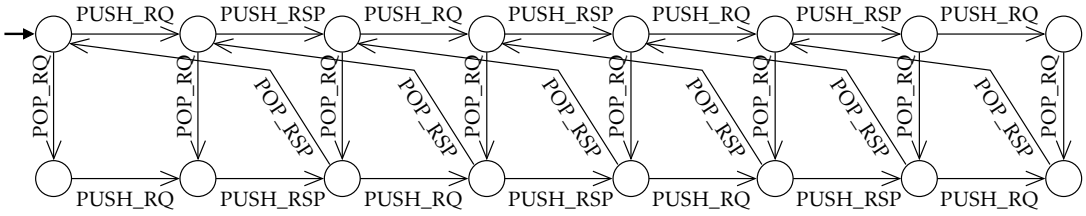


Figure 7.10: LTS of a 3-places FIFO queue with push and pop interfaces

virtual queue composed of a one-place push queue and a two-places pop queue, and with credit threshold set to one. This virtual queue is branching equivalent to a three-places FIFO queue. We depict the LTS of a three-places FIFO queue that does not distinguish data elements from signal elements in figure 7.10.

Increasing the credit threshold modifies the behavior of the virtual queue. When hiding all internal transitions, one can observe that elements are always managed following a FIFO scheme, but, for an external observer, the size of the virtual queue seems to vary.

For instance consider a virtual queue composed of a one-place push queue, a two-places pop queue and with credit threshold set to two. If the virtual queue would behave like a FIFO queue, it would be always able to contain three elements, which is actually not the case. Consider the following execution:

- initially, the virtual queue is empty
- three elements are inserted in the virtual queue and the fourth insertion is blocked (to accept the three insertions, the push queue forwards the elements to the pop queue and decreases its credit counter until it is equal to zero)
- an element is withdrawn from the pop queue

At this point a standard FIFO queue is able to grant the push operation, but the virtual queue does not. Indeed, the pop queue has a free place, but does not authorize the push queue to insert new elements, by sending a credit message. For an external observer, the virtual queue has now a maximum size of two elements.

When a second element is withdrawn from the pop queue, a credit message is send from the pop queue authorizing the push queue to send two additional elements. For an external observer, the virtual queue has anew a maximum size of three elements.

We call *abstracted virtual queue* the quotient, according to the branching bisimulation, of a virtual queue after hiding of internal transitions. The behavior of an abstracted virtual queue only depends on the total size of the virtual queue (cumulative size of the push queue and the pop queue) and on the credit threshold. As a consequence, one can say that there exist virtual queues with different size of push queues and different size of pop queues that are branching equivalent after hiding of internal transitions. For instance, a virtual queue composed of a

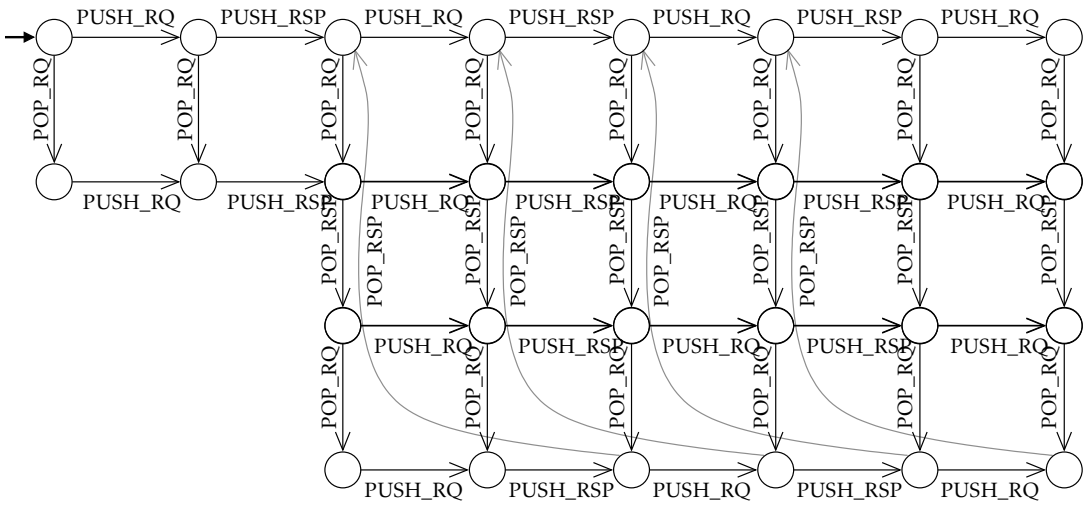


Figure 7.11: LTS of a 3-places abstracted virtual queue with credit threshold set to two

one-place push queue, a three-places pop queue and with threshold set to two is branching equivalent, after hiding of internal transitions, to a virtual queue composed of a two-places push queue, a two-places pop queue and with threshold set to two.

We depict the LTS of a three-places abstracted virtual queue that does not distinguish data elements from signal elements, with credit threshold set to two in figure 7.11.

We can study the model size of an abstracted virtual queue according to the number of elements it can contain and to the credit threshold. We depict the state space of the abstracted virtual queue according to the credit threshold for different size of the abstracted virtual queue in figure 7.12. One can first remark that, for a given size of the abstracted virtual queue, the state space of its model increases with the credit threshold. Between the two extremal configurations of the credit threshold (credit set to one and credit set to the size of the abstracted virtual queue minus one), the state space of the model almost doubles. The second remark concerns the state space of the model according to the size of the abstracted virtual queue. The state space of the models in figure 7.12 is depicted with a logarithmic scale (base 2). One can see that the state space of the model of the virtual queue almost doubles each time the size of the queue is increased by one. This is the consequence of the distinction between data and signals in the queue. The state space of the model of an abstracted virtual queue follows an exponential growth with respect to the size of the queue.

7.2.5 Complex Virtual Queue

We construct a detailed abstraction of a virtual queue in the xSTream architecture by successive refinement of a simple virtual queue, as depicted in figure 7.13. At each refinement step, we compare the new model to an abstracted virtual queue model.

1st Refinement : Addition of a Multiplexer on the Data Path

The first refinement of the virtual queue consists in the addition of a multiplexer between the push queue and the pop queue on the data path, as depicted in figure 7.13*(1)*. The credit

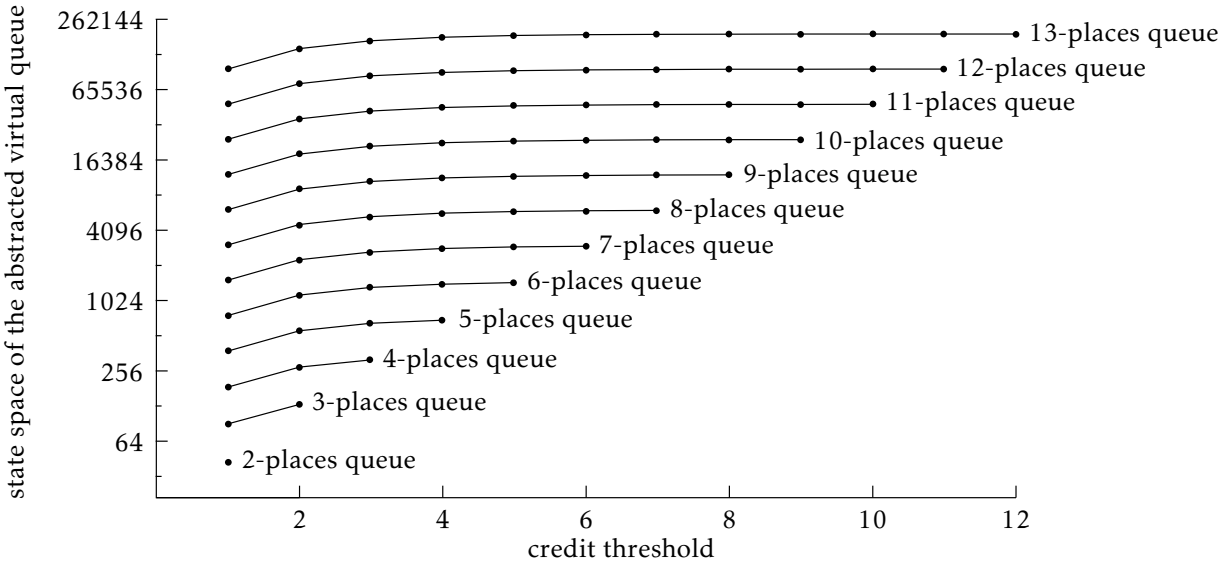


Figure 7.12: State space of an abstracted virtual queue with respect to the credit threshold

interfaces of push and pop queues remain directly connected.

A multiplexer presents two input interfaces (I_1 and I_2) and one output interface (O). The input interface I_1 of the multiplexer is connected to the output interface O of the push queue. The output interface O of the multiplexer is directly connected to the input interface I of the pop queue. The input interface I_2 of the multiplexer is left unconnected (for generation of the model, this means that the multiplexer is never able to interact on this interface).

When only the interface I_1 is connected, the functional behavior of the multiplexer is the following: it receives a request on I_1 with a pop queue identifier and an element (data or signal), then it forwards it on its output interface O (with the pop queue identifier and the element) and waits for an acknowledgment on O; finally it acknowledges on I_1.

When the two input interfaces I_1 and I_2 are connected, the multiplexer is able to receive request on I_1 and I_2 at any time but can forward them through its output O only if the last request on O has been acknowledged. Notice that a multiplexer does not buffer elements: it just forwards them.

When abstracting from internal transitions, the model of a virtual queue with a multiplexer on data path is branching equivalent to an abstracted virtual queue (configured with a size equal to the cumulative size of push and pop queues and a credit threshold equal to the one of the pop queue).

2nd Refinement : Addition of a Demultiplexer on the Data Path

The second refinement is the addition, on the data path, of a demultiplexer between the multiplexer and the pop queue, as depicted in figure 7.13*(2)*.

A demultiplexer present one input interface (I) and two output interfaces (O_1 and O_2). The output interface O of the multiplexer is connected to the input interface I of the demultiplexer. The output interface O_1 of the demultiplexer is connected to the input interface I of the pop queue. The second output interface O_2 of the demultiplexer is let unconnected (for generation

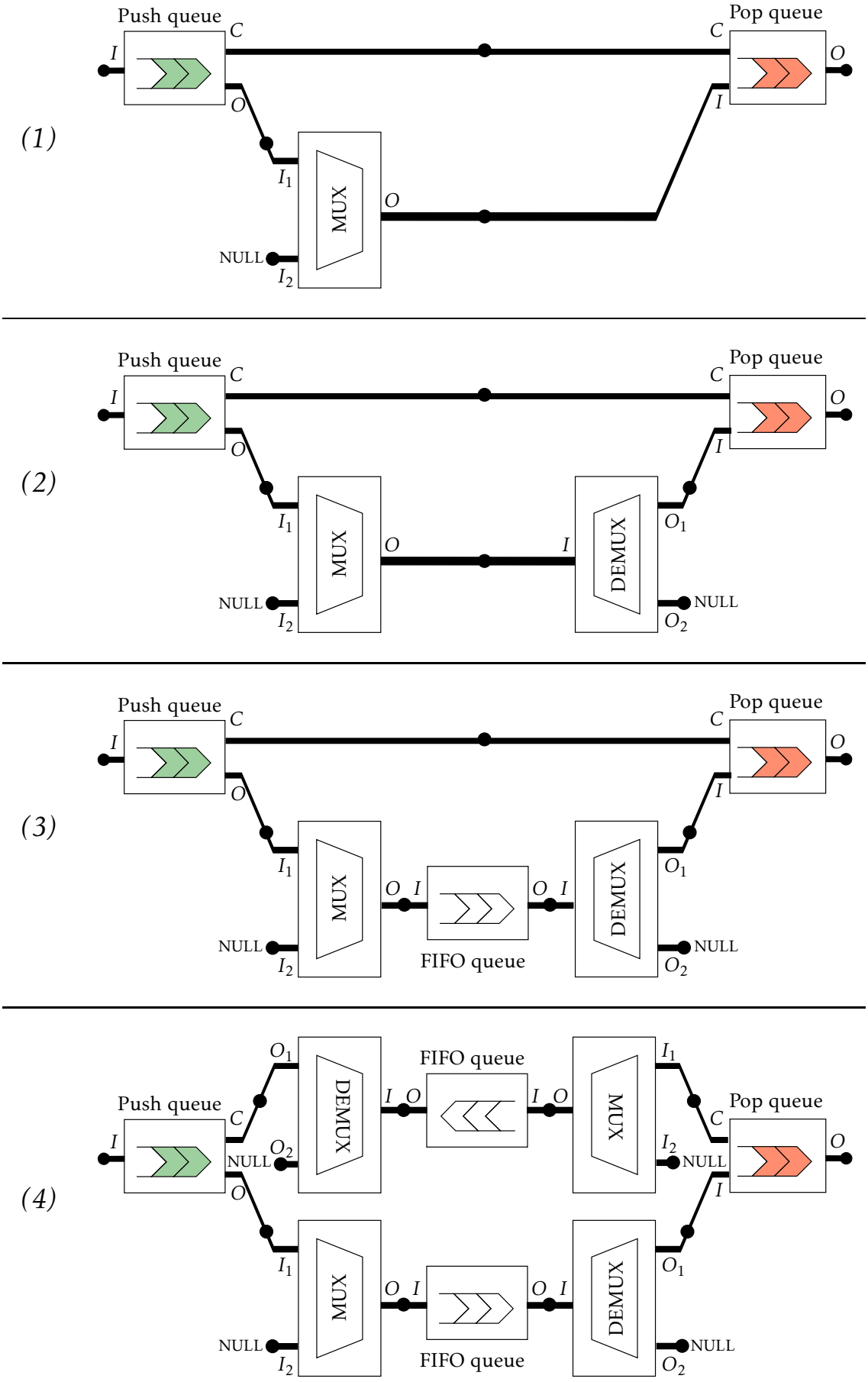


Figure 7.13: Detailed virtual queue constructed by iterative refinement of a simple virtual queue

of the model, this means that the demultiplexer is never able to interact on this interface).

The functional behavior of the demultiplexer is the following: the demultiplexer receives a request on its input I with a pop queue identifier and an element (data or signal) ; it forwards the element on the correct output interface according to the pop queue identifier ; it waits for an acknowledgment on the same output interface ; finally, it acknowledges on its input interface I. As for a multiplexer, a demultiplexer does not buffer elements.

When abstracting from internal transitions, the model of virtual queue with multiplexer and demultiplexer on data path also remains branching equivalent to an abstracted virtual queue (configured with a size equal to the cumulative size of push and pop queues and a credit threshold equals to the one of the pop queue).

3rd Refinement : Addition of a NoC Abstraction on the Data Path

The third refinement consists in modeling the data path in the NoC. We insert a standard FIFO queue between the multiplexer and the demultiplexer of the data path, as depicted in figure 7.13*(3)*. This FIFO queue presents one input interface (I) and one output interface (O). The output interface O of the multiplexer is connected to the input interface I of the FIFO queue. The output interface O of the FIFO queue is connected to the input interface I of the demultiplexer. The elements stored by this FIFO queue are pairs composed of a pop queue identifier and a data or a signal. Its behavior is a simple first-in first-out behavior.

When abstracting from internal transitions, the model of virtual queue with multiplexer and demultiplexer on data path is also branching equivalent to an abstracted virtual queue (configured with a size equal to the cumulative size of push and pop queues and a credit threshold equals to the one of the pop queue).

This equivalence may be surprising: the FIFO queue used as abstraction of the NoC does not modify the size of the virtual queue. This is the consequence of the credit protocol. The number of elements the push queue can send (the push counter) is only linked to the size of the pop queue. Consequently, the push queue cannot send more elements than the pop queue can store and places on the FIFO queue abstracting the NoC are not considered as storage places.

4th Refinement : Modeling of the Credit Path

Finally the fourth refinement consists in refining the credit path between the pop queue and the push queue, similarly to the refinement of the data path. Consequently, a multiplexer, a standard FIFO queue and a demultiplexer are added on the credit path as depicted in figure 7.13*(4)*. The interfaces of those added components are the same as the ones of the components added on the data path. Only the information exchanged on the interfaces differ. Instead of a data or signal, a credit message is sent. In addition, it is associated to a push identifier in place of the pop identifier.

Anew, when abstracting from internal transitions, the model of virtual queue with both data path and credit path refined is branching equivalent to an abstracted virtual queue (configured with a size equal to the cumulative size of push and pop queues and a credit threshold equals to the one of the pop queue).

This 4th model leads to the architecture model we use in the following. Intermediate refinements has been presented to illustrate our modeling methodology. According to this 4th model,

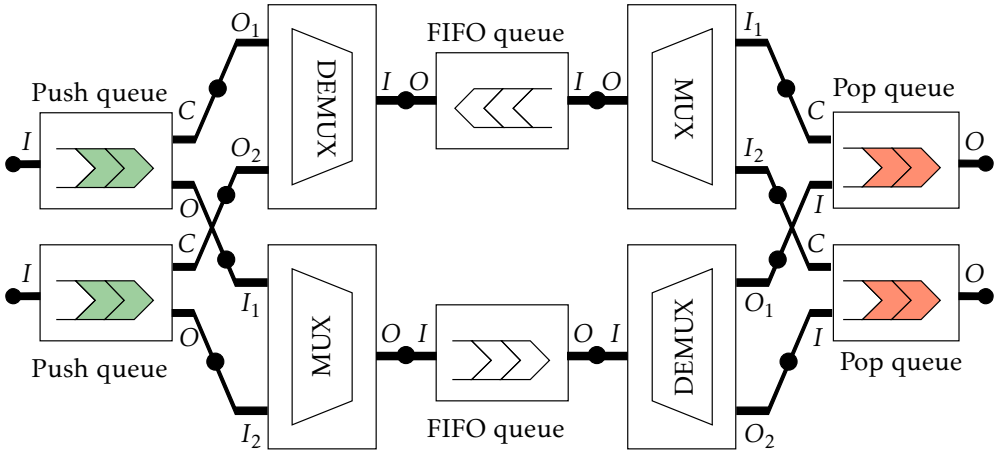


Figure 7.14: Two parallel virtual queues sharing resources on the NoC

we can conclude that an isolated virtual queue (i.e., there is a single virtual queue on the architecture) should behave like an abstracted virtual queue. In other words, the architecture does not impact the functional behavior of a virtual queue, if it is isolated.

7.2.6 Two Parallel Virtual Queues Sharing Resources on the NoC

One can now study the same virtual queue, but with interaction with other virtual queues. We use the same model as in the previous section (i.e., the 4th refinement depicted in figure 7.13*(4)*), and a second virtual queue, connecting its push queue to the unused interfaces of the data path multiplexer and the credit path demultiplexer and its pop queue to the unused interfaces of the data path demultiplexer and the credit path demultiplexer, as depicted in figure 7.14.

We focus on four experiments, presenting different configurations:

- experiment 7.2(a): the first virtual queue is composed of a one-place push queue, a one-place pop queue, and has its credit threshold set to one. The second virtual queue is identical to the first one. The NoC is abstracted by a one-place queue on the data path and a one-place queue on the credit path.
- experiment 7.2(b): Virtual queues are identical to the ones of experiment 7.2(a). The NoC is abstracted by a two-places queue on the data path and a two-places queue on the credit path.
- experiment 7.2(c): the first virtual queue is composed of a two-places push queue, a two-places pop queue, and has its credit threshold set to one. The second virtual queue is composed of a one-place push queue, a one-place pop queue, and has its credit threshold set to one. The NoC is abstracted by a one-place queue on the data path and a one-place queue on the credit path.
- experiment 7.2(d): the first virtual queue is composed of a two-places push queue, a two-places pop queue, and has its credit threshold set to one. The second virtual queue is identical to the first one. The NoC is abstracted by a two-places queue on the data path and a two-places queue on the credit path.

We want to compare those four experiments to abstracted virtual queues. To that purpose, in each model, we hide all actions but the interfaces of push and pop operations. Results are

Experiments			7.2(a)	7.2(b)	7.2(c)	7.2(d)
	1st virtual queue	push queue length	1	1	2	2
		pop queue length	1	1	2	2
		credit threshold	1	1	1	1
	2nd virtual queue	push queue length	1	1	1	2
		pop queue length	1	1	1	2
		credit threshold	1	1	1	1
	NoC	data path length	1	2	1	2
		credit path length	1	2	1	2
largest intermediate model		number of states	273024	305280	3374208	66399120
		number of transitions	1520736	1693536	20936728	450101456
minimized model		number of states	1764	1764	7812	34596
		number of transitions	6636	6636	30108	136524
bisimilar to	1st abstracted virtual queue (in parallel)	queue length	2	2	4	4
		credit threshold	1	1	1	1
	2nd abstracted virtual queue	queue length	2	2	2	4
		credit threshold	1	1	1	1

Figure 7.15: Results of the construction of two parallel virtual queues sharing resources on the NoC

depicted in figure 7.15. In the construction of the models for the four experiments, we present the size of the largest intermediate model used during composition and the size of the resulting minimized model.

One can firstly analyze size of the different models. For experiment 7.2(d), we can see that the largest intermediate model used in compositions has more than sixty millions of states and four hundred fifty millions of transitions. However, after minimization, all our experiments result in a rather small model of up to thousands of states. For each experiment, we compare the minimized model obtained to a model obtained with parallel abstracted virtual queues. More precisely, we compared:

- minimized model of experiment 7.2(a) to the parallel composition of two abstracted virtual queues of total length two and credit threshold set to one
- minimized model of experiment 7.2(b) to the parallel composition of two abstracted virtual queues of total length two and credit threshold set to one (i.e., as for experiment 7.2(a))
- minimized model of experiment 7.2(c) to the parallel composition of an abstracted virtual queue of total length four and credit threshold set to one and an abstracted virtual queue of total length two and credit threshold set to one
- minimized model of experiment 7.2(d) to the parallel composition of two abstracted virtual queues of total length four and credit threshold set to one

For all our experiments, we obtained the branching equivalence between the constructed minimized model and the model obtained by parallel composition of abstracted virtual queues. Consequently, virtual queues sharing resources on the NoC should behave like abstracted virtual queues in parallel, i.e., although virtual queues share resources, their functional behavior remains the one of an isolated queue.

7.3 Performance Measures for the xSTream architecture

Contrary to functional verification, one needs models of application (i.e., models of filters) to investigate the performance of a system. Indeed, only closed models can be tackled by our performance methodology (cf section 5.2.3). To that purpose, we call *producer* the abstraction of an application inserting elements in the communication architecture and *consumer* the abstraction of an application consuming elements from the communication architecture. Naturally, a producer is linked to a push queue and a consumer is linked to a pop queue.

The computation of all the results presented in this section took only few hours, the most time consuming part of the performance evaluation being the modeling of systems.

7.3.1 Study of a Simple Virtual Queue

We focus on the performance evaluation of a simple virtual queue, for which the functional model was presented in section 7.2.4. Our goal is to study the influence of the length of a virtual queue on its performance, disregarding the influence of the credit threshold of the pop queue composing the simple virtual queue. Thus, the credit threshold is arbitrarily fixed to one. In particular, we want to underline the gain provided by the modeling of delays considering their probabilistic distribution instead of just considering their average value: for a given delay in the model, using different probabilistic distributions with the same average value may lead to different performance results.

For performance evaluation, delays are inserted in functional models in a constraint-oriented way, as presented in chapter 6. Additionally, the push interface of the push queue is connected to a producer that performs push operations and the pop interface of the pop queue is connected to a consumer that performs pop operations. To study the impact of the length of the virtual queue on performance, we consider that elements are produced by bursts:

- the producer emits 20 elements, one element every two time units (i.e., the time between the PUSH_RSP of an element and the PUSH_RQ of the next one is equal to two time units). Then, the producer waits for 102 time units before looping.
- the consumer tries to withdraw an element every eight time units, i.e., the time between the POP_RSP of an element and the POP_RQ of the next one is equal to eight.

In a first section, we illustrate the influence of the respective sizes of the push and the pop queues of a virtual queue on performance results. By adjusting correctly the different delays, we can consider a model where the respective sizes of the push and pop queues have no impact on the performance results. In this case, only the total size of the virtual queue (sum of the push queue size and the pop queue size) has an impact on the performance results. We use such a model in a second section to underline the advantages of considering time distribution for delays instead of average values.

Influence of Push and Pop Queue Sizes

Functionally, we verified that the behavior of a virtual queue only depends on its size (i.e., size of the push queue plus size of the pop queue) and the credit threshold. Whatever the individual size of push and pop queues, only their cumulated size has an impact on the behavior of the queue. For the performance, this property is not always satisfied. For instance, if the throughput at the interface between the push and pop queues is lower than the throughput of

Experiments	Size of the push queue
7.3.1(a)	1
7.3.1(b)	2
7.3.1(c)	3

Table 7.1: Configuration for experiments 7.3.1(a), 7.3.1(b) and 7.3.1(c)

the producer, the push queue may be full, blocking push operations, although there are free places in the pop queue. Using different size for the push queue, we illustrate this problem with three experiments summarized in table 7.1.

Delays inserted in the models of experiments 7.3.1(a), 7.3.1(b) and 7.3.1(c), are presented in table 7.3.1. Notice that the interface delay D_{if}, corresponding to the time needed to insert an element from the push queue into the pop queue, is not fixed but follows a probabilistic distribution, modeling the time needed to pass through the NoC.

According to the parameters used in the model, we can compute the theoretical average throughput of the producer, Thr_{prod}, and of the consumer, Thr_{cons}, i.e., the throughput reached when there are never blocked push or pop operations. Thr_{prod} (resp. Thr_{cons}) is equal to the average time between two produced (resp. consumed) elements, plus the average time needed to perform a push operation (resp. a pop operation). Additionally, we can compute the theoretical average throughput at the interface between the push queue and the pop queue, Thr_{if}, which corresponds to the inverse of the average of the delay D_{if}. Consequently, we have:

- Thr_{prod} is equal to one element every eight time units
- Thr_{cons} is equal to one element every 9.5 time units
- Thr_{if} is equal to one element every seven time units

Theoretically, because Thr_{prod} is higher than Thr_{cons}, the queue should, on average, be full. Moreover, the average production throughput should not exceed the average consumption throughput: the maximum reachable production throughput is equal to Thr_{cons}. Concerning Thr_{if}, because it is higher than Thr_{prod}, the performance of the virtual queue should not be impacted by the interface delay: on average, the push queue forwards elements to the pop queue faster than they are produced.

Actually, however, the performance of the virtual queue is affected by the interface delay because elements are produced by bursts, as illustrated in figure 7.16. For experiments 7.3.1(a), 7.3.1(b) and 7.3.1(c), we study the push operation latency, i.e., the time effectively elapsed between a PUSH_RQ and a PUSH_RSP. The push operation latency corresponds to the time physically needed to process the push operation (i.e., the push operation delay), plus the time the producer was blocked before the operation is processed (i.e., the time before a place is available in the push queue). When the push operation latency is close to the push operation delay, push operations are seldom blocked because the push queue is full. Conversely, when the push operation latency is much higher than the push operation delay, push operations are often blocked.

In figure 7.16, the average latency of the push operation is depicted for experiments 7.3.1(a), 7.3.1(b) and 7.3.1(c), as a function of the size of the virtual queue. Notice that experiment 7.3.1(b) is not depicted for a virtual queue size of two and experiment 7.3.1(c) is not depicted for a virtual queue size of two and three. Indeed, the size of the virtual queue is at least equal to the size of the push queue plus one.

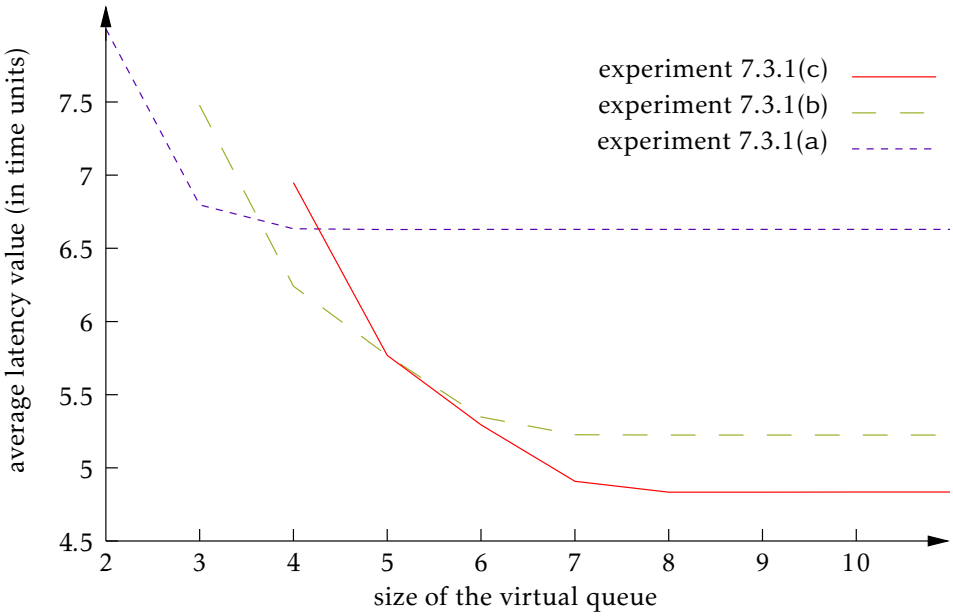


Figure 7.16: Push operation latency. The performance of a virtual queue may depend on respective sizes of the push and pop queue, and not only on their cumulated size

One can first remark that the three experiments do not provide the same results, which could be surprising since the three virtual queues are functionally equivalent and inserted delays are the same. It is directly linked to the size of the push queue: the larger the push queue, the smaller the probability for a push operation to be blocked.

In addition, we can note that a minimum value is reached, which does not decrease anymore when the pop queue size is increased. For all three experiments, the minimum value for the push operation latency is far from the push operation delay (equal to one time unit on average) that is a lower bound for the push operation latency. This implies that the push queue is often blocked in all three experiments, which confirms that, on average, the virtual queue is full.

We can also see that the virtual queue of experiment 7.3.1(a) is more efficient than the virtual queue of experiment 7.3.1(b) for a size of three. Similarly, the virtual queue of experiment 7.3.1(a) is more efficient than the virtual queue of experiment 7.3.1(c) for a size of four. Although push queues in experiments 7.3.1(b) and 7.3.1(c) are larger than the push queue in experiment 7.3.1(a), the virtual queues of experiments 7.3.1(b) and 7.3.1(c) with a size of three and four are handicapped by their limited pop queues (they have only one place).

As a consequence, we can conclude that the size of the push queue and the size of the pop queue composing a virtual queue may influence its performance. The performance of a virtual queue is thus not only linked to the total size of its push queue and its pop queue.

Advantages of Probabilistic Distributions of Time in Models

Considering the conclusion of the previous section, we could guess that the influence of the size of the push queue and the pop queue on the performance of the virtual queue is the consequence of an insufficient throughput $\mathrm{Thr}_{\mathrm{if}}$ at their interface: a push operation can be blocked because the push queue is full, although there are free places in the pop queue. For a

Symbol	Distribution		Name and description
	Value	Prob.	
D_{push}	1 t.u.	1	*push operation delay*: time physically needed to insert an element in the push queue
D_{cdt}	2 t.u.	1	*credit delay*: time needed in the push queue to take into account an incoming credit message
D_{if}	6 t.u. 7 t.u. 8 t.u.	0.3 0.4 0.3	*interface delay*: time needed to insert an element from the push queue into the pop queue
D_{pop}	1 t.u. 2 t.u.	0.5 0.5	*pop operation delay*: time physically needed to withdraw an element from the pop queue the push queue into the pop queue

t.u. = time unit

Table 7.2: Delays inserted in the virtual queue model for experiments 7.3.1(a), 7.3.1(b) and 7.3.1(c)

Experiments	Distribution of D_{push}	
	Value	Prob.
7.3.1(d)	2 t.u. 3 t.u.	0.5 0.5
7.3.1(e)	1 t.u. 2 t.u. 3 t.u. 4 t.u. 5 t.u. 6 t.u.	0.5 0.1 0.1 0.1 0.1 0.1
7.3.1(f)	1 t.u. 16 t.u.	0.9 0.1

t.u. = time unit

Table 7.3: Configuration for experiments 7.3.1(d), 7.3.1(e) and 7.3.1(f)

high value of the throughput Thr_{if}, i.e., when there are never blocked push operations whilst the pop queue has free places, the performance of the virtual queue should only depend on the cumulative size of the push queue and of the pop queue. This claim can be verified: the throughput Thr_{if} has no influence if two equally sized virtual queues, with identical delays, but with different sizes for their respective push and pop queues, should leads to the same performance result.

In this section, we parameterize virtual queues such as to minimize the influence of the throughput Thr_{if} of the interface between the push queue and the pop queue on the performance results. We configure the throughput Thr_{if} with the highest possible value, i.e., with a delay D_{if} equal to one time unit (with probability one).

We study three different virtual queues that have the same push operation delay (D_{push}) on average. The distribution of D_{push} for each experiment is given in table 7.3.

The considered distribution for the push operation delay can be seen as the timed behavior

of the push interface of a push queue, abstracting its backlog mechanism (i.e., when elements are stored in memory and not in the hardware queue): insertion of an element takes either a small time value, which corresponds to a physical insertion in the queue, or a larger time values, which corresponds to an insertion into the backlog memory.

The three virtual queues of experiments 7.3.1(d), 7.3.1(e), and 7.3.1(f), are composed of a one-place pop queue with a credit threshold set to one. They are also connected to identical producers and identical consumers that are the ones presented at the beginning of the section. The delays inserted (excepted D_{push}) are the same for all the experiments and are presented in table 7.3.1.

With those parameters, the time between two produced elements is equal to the push operation delay plus the time the producer waits before producing the next element. The maximum throughput for the producer is reached when the queue is never full, i.e., a push operation is never blocked. On average, the throughput of the producer is equal to one element every 9.5 time units. Similarly, the time between two consumed elements is equal to the pop operation delay plus the time the consumer waits before consuming the next element. The maximum theoretical throughput of the consumer is reached when the queue is never empty, i.e., a pop operation is never blocked. On average, the throughput of the consumer is equal to one element every nine time units. Because the throughput of the consumer is greater than the throughput of the producer, the effective maximum throughput the consumer can reach, on average, is the throughput of the producer, i.e., one element every 9.5 time units.

For the three experiments, we study two performance measures for different sizes of the virtual queue:

- the latency of a push operation, i.e., the time elapsed between a PUSH_RQ and the corresponding PUSH_RSP
- the throughput of the virtual queue, i.e., the throughput at the pop interface of the pop queue.

Those two measures can be studied using latencies, as defined in chapter 3. The throughput measure corresponds to the study of the latency defined by the time elapsed between two POP_RQ. We depict those two measures as functions of the size of the virtual queues of experiments 7.3.1(d), 7.3.1(e) and 7.3.1(f) in figure 7.17.

A first observation is that, whatever the distribution of the push operation delay and the size of the virtual queue, the latency associated to the throughput of the virtual queue is correlated to the push operation latency. When the size of the virtual queue increases, the latency associated to the throughput of the virtual queue decreases simultaneously with the push operation latency.

For small sizes, the virtual queues of experiments 7.3.1(d) and experiment 7.3.1(e) have almost the same performance, but performs better than the one of experiment 7.3.1(f), although the distributions of D_{push} have the same average value in all three experiments. This is explained by the way elements are inserted. In experiment 7.3.1(d), elements are inserted at a quite stable rate. The throughput of insertion is also quite stable and lesser than the throughput at the interface between the push queue and the pop queue. Consequently the push queue has time to forward elements to the pop queue and should not be often full. For experiment 7.3.1(e), elements are inserted at a high rate with a high probability (the time to insert an element is equal to one time unit with a probability 0.5) and at lower rates with lower probabilities. The push queue has consequently a higher probability to be full, and thus to block push operations. The experiment 7.3.1(f) emphasizes the behavior of experiment 7.3.1(e): elements

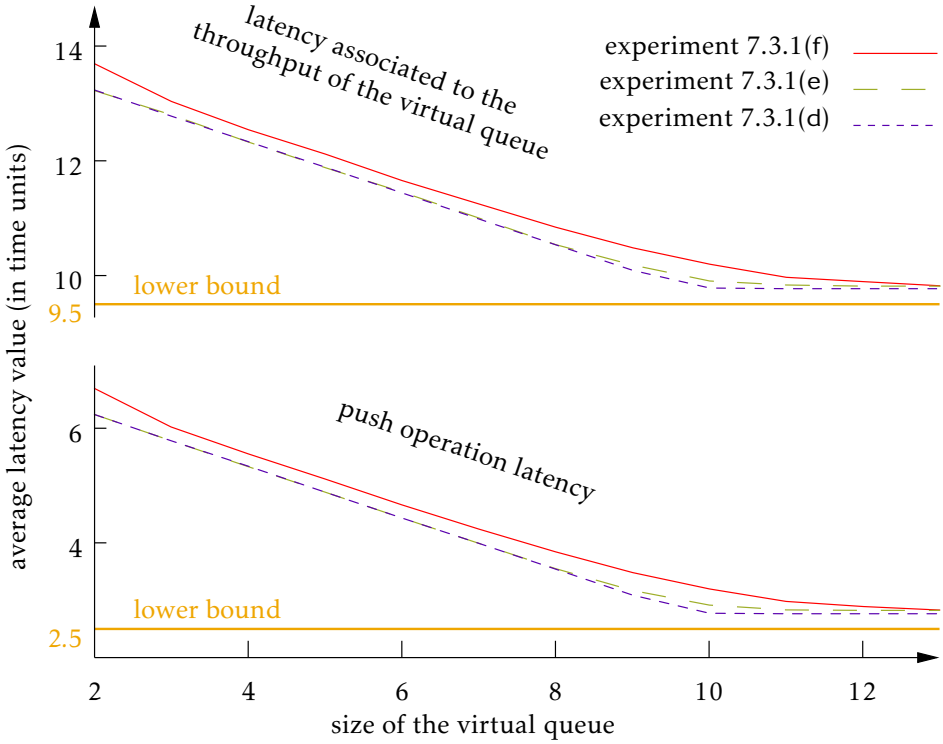


Figure 7.17: Push operation latency and latency associated to the virtual queue throughput

are inserted at a high rate with a very high probability (the time to insert an element is equal to one time unit with a probability 0.9) and at a very low rate with a very small probability.

Nevertheless, when their sizes are increased, virtual queues of the three experiments provide similar performance: in all three experiments the push operation latency and the latency associated to the throughput of the virtual queue converge to their lower bounds.

As a conclusion we can say that it is important to consider distributions instead of only average values for delays inserted in our models. As we have seen, for the same delay on average, and for small sizes, a virtual queue may be more or less efficient. The influence of the considered distributions, comparing to their average value, diminishes as the size of the virtual queue is increased.

This conclusion is interesting replacing the model in the industrial context of the xSTream architecture. We saw that according to the delays induced by the architecture, virtual queues are more or less efficient. The study of the architecture with accurate delays is an interesting information for programmers. Indeed, a programmer can know whether optimizations on its applications (to improve their performance) will be supported by the architecture.

7.3.2 Study of Two Parallel Virtual Queues Sharing Resources on the NoC

In this section, we address the performance evaluation of two parallel virtual queues, sharing resources on the NoC. We enrich the functional model presented in section 7.2.6 with time information. We focus on the impact of the credit protocol in terms of performance of the system.

Symbol	Distribution		Name and description
	Value	Prob.	
D_{cdt}	1 t.u. 2 t.u.	0.5 0.5	*credit delay*: time needed in the push queue to take into account an incoming credit message
D_{if}	1 t.u.	1	*interface delay*: time needed to insert an element from the push queue into the pop queue
D_{pop}	1 t.u.	1	*pop operation delay*: time physically needed to withdraw an element from the pop queue the push queue into the pop queue

t.u. = time unit

Table 7.4: Delays inserted in the virtual queue model for experiments 7.3.1(d), 7.3.1(e) and 7.3.1(f)

Experiments	Credit protocol implemented ?	Length of the pipeline of one-place FIFO queues on the data path
7.3.2(a)	YES	1
7.3.2(b)	NO	1
7.3.2(c)	YES	3
7.3.2(d)	NO	3

Table 7.5: Configuration for experiments 7.3.2(a), 7.3.2(b), 7.3.2(c), and 7.3.2(d)

To model the time needed to pass through the NoC on data path, we do not use a simple FIFO queue, but a pipeline of one-place FIFO queues. Functionally, a pipeline of n one-place FIFO queues is branching bisimilar to the n-places FIFO queue depicted in section 7.2.5. The functional model of a pipeline of three one-place FIFO queues is given in section E.4 of appendix E. A pipeline of one-place FIFO queues to abstract the timed behavior of the NoC is suited, because we can insert a delay for each one-place FIFO queue in the pipeline.

We focus in the study of two different models, the former implementing the credit protocol, and the latter not implementing it:

- the model depicted in figure 7.18 corresponds to the model of section 7.2.6 enriched with time information (the FIFO queue on the data path is replaced by a pipeline of one-place FIFO queues).
- the model depicted in figure 7.19 is similar to the first one, but the credit protocol is not modeled. In this second model, push and pop queues do not implement the credit protocol, and there is consequently no credit path. However, there is still the data path, with a multiplexer, a NoC abstraction (i.e., a pipeline of one-place FIFO queues), and a demultiplexer.

The two virtual queues are composed of a two-places push queue and a two-places pop queue. Let *virtual queue* 1 be the first virtual queue and *virtual queue* 2 be the second virtual queue. *push queue* 1 (resp. *push queue* 2) and *pop queue* 1 (resp. *pop queue* 2) denote the push queue and the pop queue composing virtual queue 1 (resp. virtual queue 2).

The abstraction of the NoC on the credit path is a one-place FIFO queue. We consider two different lengths for the pipeline of FIFO queues abstracting the NoC on the data path. Consequently, we have four different experiments that are summarized in table 7.5.

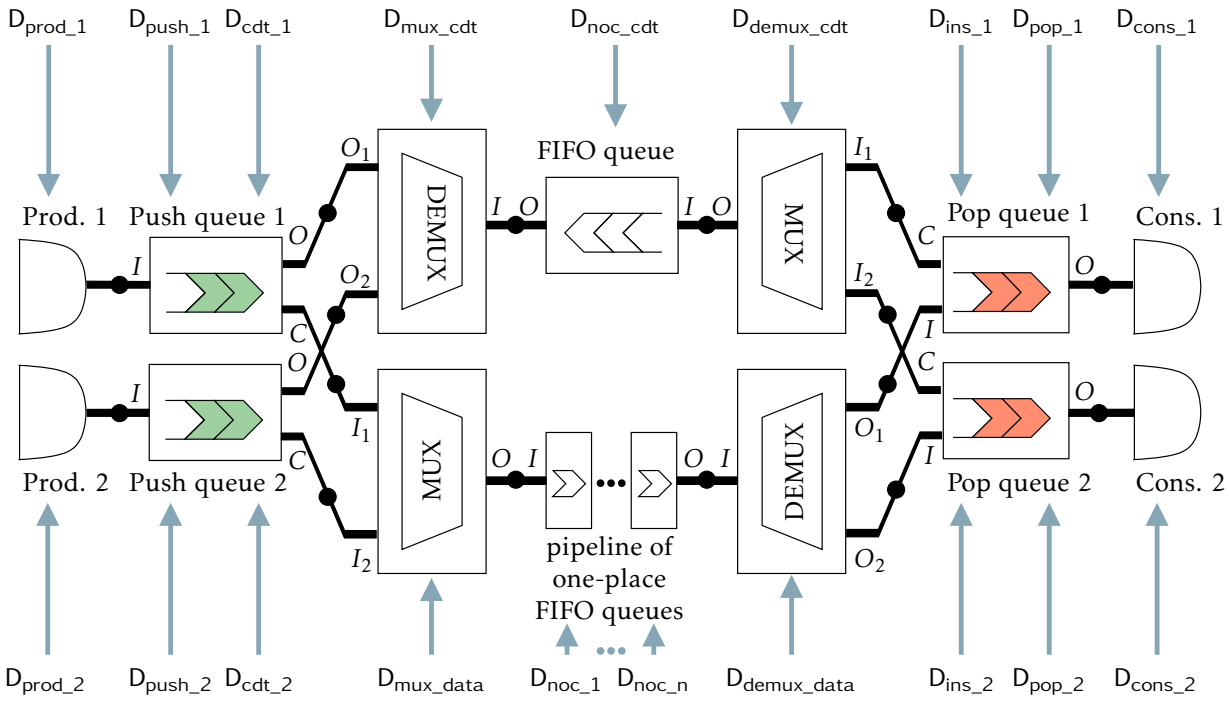


Figure 7.18: Timed model implementing the credit protocol

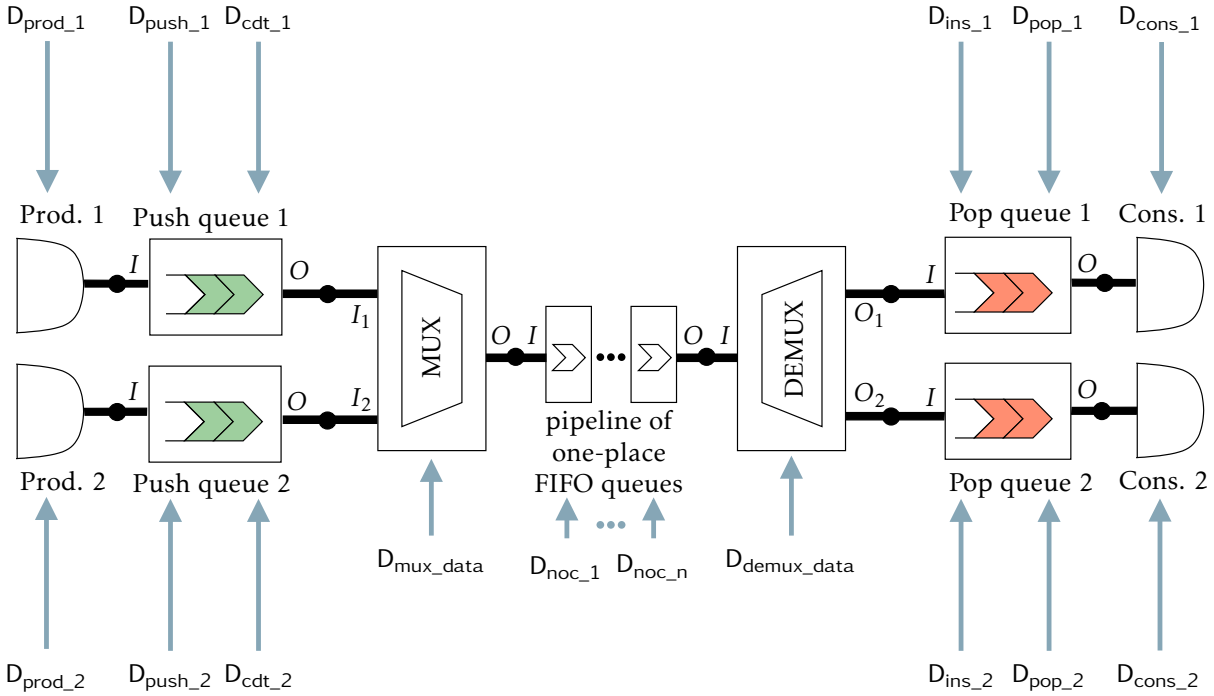


Figure 7.19: Timed model not implementing the credit protocol

	Symbol	Distribution		Name and description
		Value	Prob.	
experiments 7.3.2(a) and 7.3.2(b)	D_{noc_1}	1 t.u.	1	*Data NoC delay* 1: insertion delay of an element in the first one-place FIFO queue of the pipeline abstracting the NoC on the data path
	D_{noc_2}	1 t.u.	1	*Data NoC delay* 2: insertion delay of an element in the second one-place FIFO queue of the pipeline abstracting the NoC on the data path
	D_{noc_3}	1 t.u. 2 t.u. 3 t.u.	0.4 0.3 0.3	*Data NoC delay* 3: insertion delay of an element in the third one-place FIFO queue of the pipeline abstracting the NoC on the data path
exp. 7.3.2(c) and 7.3.2(d)	D_{noc_1}	3 t.u. 4 t.u. 5 t.u.	0.4 0.3 0.3	*Data NoC delay* 1: insertion delay of an element in the one-place FIFO queue abstracting the NoC on the data path

t.u. = time unit

Table 7.6: Delays inserted in the NoC

To compare performances of models where the length of the pipeline of one-place FIFO queues on the data path is one, to models where the length of the pipeline of one-place FIFO queues on the data path is three, we set delays such as to have the same probabilistic distribution for the time to pass through the NoC, i.e., three time units with a probability 0.4, four time units with a probability 0.3 and five time units with a probability 0.3. The delays inserted in the abstraction of the NoC are presented in table 7.6.

Other delays inserted in the models are presented in the upper part of table 7.7. Notice that all those delays, inserted in the models, are deterministic.

To investigate the influence of the credit protocol with respect to the sharing of the NoC, we consider that virtual queue 1 (resp. virtual queue 2) is connected to a producer, called *producer* 1 (resp. *producer* 2), and to a consumer, called *consumer* 1 (resp. *consumer* 2). We study the throughput of virtual queue 1 when the average consumption rate of consumer 2 decreases.

In producers and consumers, delays are inserted, corresponding to the time elapsed between the response of an operation (PUSH_RSP or POP_RSP) and the request of the next one (PUSH_RQ or POP_RQ). The delays inserted in producer 1, producer 2 and consumer 1 are presented in the lower part of table 7.7.

For consumer 2, the delay D_{cons_2} between a POP_RSP of an element withdrawal and the POP_RQ of the next one will vary in our experiments: it will be set to i time units with a probability one, i varying from one to 40 time units. We study the average latency between two POP_RQ in pop queue 1, which is the inverse of the throughput of virtual queue 1. Results for experiments 7.3.2(a), 7.3.2(b), 7.3.2(c) and 7.3.2(d) are depicted in figure 7.20.

Independently from the length of the pipeline of one-place FIFO queues abstracting the NoC, we can see that, for small values of D_{cons_2}, the implementation of the credit protocol

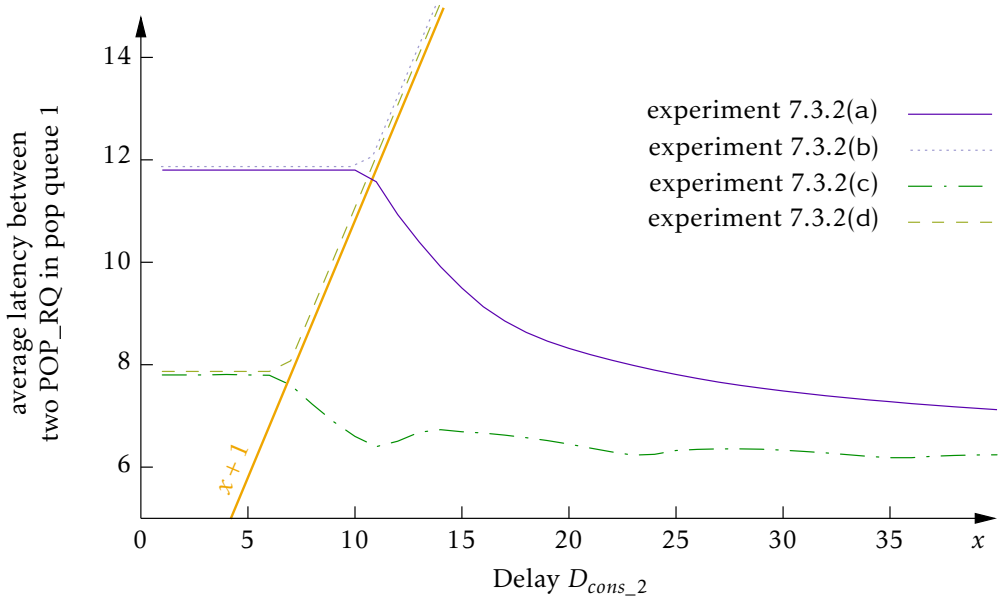


Figure 7.20: average latency between two POP_RQ in push queue 1

has no impact on the latency between two POP_RQ in push queue 1. This observation is the consequence of the configuration of the credit protocol: the credit threshold is set to one in both pop queues, and the time for a credit message from a pop queue to a push queue is small (comparing to the time for an element to pass through the NoC on data path).

With the credit protocol (experiments 7.3.2(a) and 7.3.2(c)), notice that when D_{cons_2} increases, i.e., the throughput of consumer 2 decreases, the average latency between two POP_RQ in pop queue 1 decreases, i.e., the throughput of virtual queue 1 increases. Indeed, when the throughput of consumer 2 decreases, push queue 2 less often inserts elements in the shared FIFO queue on the data path. There is consequently less contention between the two virtual queues. The throughput of virtual queue 1 increases, as D_{cons_2} increases, and seems to converge to a lower bound that should correspond to the throughput of virtual queue 1 when virtual queue 2 is not used.

Finally, without the credit protocol (experiments 7.3.2(b) and 7.3.2(d)), we can see that the average latency between two POP_RQ in pop queue 1 increases linearly with D_{cons_2}, i.e., the throughput of virtual queue 1 decreases as the consumption rate on virtual queue 2 decreases. As D_{cons_2} is increased, the production rate of producer 2 becomes greater than the consumption rate of consumer 2. On average, pop queue 2 is consequently often full, and elements of virtual queue 2 may block the NoC for elements of virtual queue 1. Indeed, elements of virtual queue 2, at the head of the shared FIFO queue on the data path, may be waiting for a free place in pop queue 2.

As a conclusion, models depicted in figure 7.18 and 7.19 allow to estimate the penalty induced by the use of the credit protocol, and underline its utility to avoid interactions between virtual queues that would impact their performances. In addition to the assurance that there will not be deadlocks (as shown by functional verification), the credit protocol ensures that the performance of an application is not deeply impacted from the performance of others running in parallel.

Symbol	Distribution		Name and description
	Value	Prob.	
D_{push_1}	2 t.u.	1	*push operation delay* 1: time physically needed to insert an element in push queue 1
D_{push_2}	2 t.u.	1	*push operation delay* 2: time physically needed to insert an element in push queue 2
D_{cdt_1}	1 t.u.	1	*credit delay* 1: time needed in push queue 1 to take into account an incoming credit message
D_{cdt_2}	1 t.u.	1	*credit delay* 2: time needed in push queue 2 to take into account an incoming credit message
D_{ins_1}	1 t.u.	1	*insertion delay in pop queue* 1: time physically needed to insert an element in pop queue 1
D_{ins_2}	1 t.u.	1	*insertion delay in pop queue* 2: time physically needed to insert an element in pop queue 2
D_{pop_1}	1 t.u.	1	*pop operation delay* 1: time physically needed to withdraw an element from pop queue 1
D_{pop_2}	1 t.u.	1	*pop operation delay* 2: time physically needed to withdraw an element from pop queue 2
D_{mux_data}	1 t.u.	1	*multiplexer delay on data path*: time needed for the multiplexer on data path to forward an element from one of its inputs to its output
D_{demux_data}	1 t.u.	1	*demultiplexer delay on data path*: time needed for the demultiplexer on data path to forward an element from its input to one of its outputs
D_{mux_cdt}	1 t.u.	1	*multiplexer delay on credit path*: time needed for the multiplexer on credit path to forward an element from one of its inputs to its output
D_{noc_cdt}	1 t.u.	1	*insertion delay in FIFO queue of credit path*: time physically needed to insert an element in the standard FIFO queue abstracting NoC on credit path
D_{demux_cdt}	1 t.u.	1	*demultiplexer delay on credit path*: time needed for the demultiplexer on credit path to forward an element from its input to one of its outputs
D_{prod_1}	1 t.u. 2 t.u. 3 t.u.	0.5 0.3 0.2	*producer* 1 *delay*: time elapsed between a PUSH_RSP of an element insertion in push queue 1 and the PUSH_RQ of the next insertion
D_{prod_2}	1 t.u. 2 t.u. 3 t.u.	0.5 0.3 0.2	*producer* 2 *delay*: time elapsed between a PUSH_RSP of an element insertion in push queue 2 and the PUSH_RQ of the next insertion
D_{cons_1}	1 t.u. 2 t.u. 3 t.u.	0.5 0.3 0.2	*consumer* 1 *delay*: time elapsed between a POP_RSP of an element withdrawal in pop queue 1 and the POP_RQ of the next withdrawal

t.u. = time unit

Table 7.7: Delays inserted in the models of two virtual queues sharing resources on the NoC, i.e., for experiments 7.3.2(a), 7.3.2(b), 7.3.2(c), and 7.3.2(d)

Chapter 8

Related Work

In this chapter we overview published work related to formal modeling of timed and probabilistic systems and to analysis methods. We first present work on process algebras and their timed and probabilistic extensions related to Interactive Probabilistic Chains (IPC, see chapter 5). Then, we present various definitions of bisimulations for those models and discuss their properties. Finally, we present some existing techniques for performance evaluation on models obtained from those process algebras.

8.1 Process Algebras Dealing With Time and/or Probabilities

Process Algebras were introduced thirty years ago [Hoa78, Mil80] as adequate formalisms to describe parallelism and to reason about functional aspects of complex systems. They have been significantly extended over the years to express more than functional aspects according to the logical order of events. One can cite extensions leading to:

- **Timed process algebras**, such as [OS06, LL98, BB96, Sch95, HR95, NS94, ACD93, Yi91, RR86], extend process algebras by integrating real-time concepts (i.e., constant delays, timers, urgency, etc.) to express time properties.
- **Probabilistic process algebras**, such as [JLY01, DHK99, vGSS95, JS90, GcJS90, Sub87], are an extension providing probabilistic branching in addition to the nondeterministic branching of process algebras.
- **Probabilistic and timed process algebras**, such as [BA03, And00, Seg95, Han94, Han91, HJ90], combine both properties of timed process algebra and probabilistic process algebras.
- **Stochastic process algebras**, such as [Her02, HHK02, HHK$^+$00, Hil96, BDG$^+$95], extend process algebras with time delays whose durations follow exponential distributions. Because stochastic process algebras express both time and probabilities, they could be classed into the probabilistic and timed process algebras. However, they differ with respect to the parallelism of delays. In timed process algebras, time has to progress synchronously, whereas in stochastic process algebras, delays can be interleaved due to the memoryless property of exponential distributions.

Among stochastic process algebras, the recent Interactive Markov Chain (IMC) formalism [Her02] stands out by clearly separating Markovian transitions (representing a delay whose duration is exponentially distributed) from interactive transitions (based on atomic actions).

This approach has the advantage of avoiding any explicit synchronization of distributions. IPC formalism, presented in this thesis (cf. chapter 5), follows this way, clearly separating probabilistic transitions (representing a one time-step delay) from interactive transitions (based on atomic actions). Consequently, IPC formalism can be seen as a transposition of the IMC formalism in a discrete time context.

Nevertheless, due to the discrete nature of time that has to progress synchronously, IPCs are closer to timed process algebras than to stochastic process algebras. The integration of probabilistic branching classes IPCs in the category of probabilistic and timed process algebras.

By merging time and probabilities in IPCs, our aim was to base performance analysis on the study of Discrete Time Markov Chains (DTMC) by transforming IPCs into DTMCs. Thus, the IPC formalism provides a compositional way to generate DTMCs, similar to the use of IMCs to generate CTMCs compositionally. Theoretically, the continuous-time case in IMCs allows to cover the discrete time case provided by IPCs. However, we illustrated in chapter 4 problems encountered when using IMCs for performance evaluation of hardware systems, mainly the uncertainty about the error introduced by approximations into the obtained performance results.

In the remainder of this section, we first compare in more detail time characteristics and probabilistic characteristics of IPCs with existing process algebras. Time and probabilities being deeply intertwined in IPCs, it is difficult to abstract from the timed aspects of IPC to only study probabilistic aspects, and vice-versa. Consequently, our comparison finally focuses on models with both time and probabilistic aspects.

8.1.1 Time Models

To compare the time aspect of IPCs to existing formalisms, we consider the terminology employed in [NS91] to define a general model of timed system.

The first time characteristic of IPCs is their time domain. While several timed process algebras are defined on a dense time domain, such as [OS06, LL98, Seg95, Sch95, JG89, GB87, RR86], IPCs are defined over a discrete time domain as [BM01, BMU98, HR95, Han94, NS94, Han91, HJ90]. The term discrete time indicates that the advance of time is considered as a sequence of time steps (ticks). In those models, actions are also atomic and take no time. Time only progresses "between" sequences of actions. In timed process algebra, sequences of actions alternate with sequences of time progression.

The second time charecteristic of IPCs is their determinism in time. Timed process algebras require, in general, the time determinism property, which, informally, means that time progression does not influence the functional behavior of the system. In other words, after letting time progress, the first actions enabled will be always the same. Most process algebras dealing with both time and probabilities [BA03, And00, Seg95, Han94, Han91] satisfy the time determinism property, because time and probabilities are separated (a probabilistic choice cannot be processed during time progression). Because probabilistic branching is enabled when time progresses in IPCs, time determinism is not verified by IPCs. However, we could talk about time probabilistic determinism since the future in time is always defined by a probabilistic distribution. In this thesis, time determinism refers to a different property (cf. section 5.2.3 of chapter 5): we use time determinism to denote that, from any arbitrary state, all possible sequences of actions are confluent to the same state before the next time step. In other words, time determinism characterizes IPCs for which the unique cause of non-determinism is the

interleaving semantics of parallelism.

A third time characteristic of IPCs is that they may be theoretically Zeno, i.e., an infinite sequence of actions may occur within a finite time. The non-Zeno property is prohibited in several timed process algebras [Han94, BB91, MT90]. In the discrete-time and generative-reactive process algebra presented in [BA03], processes are obviously non-Zeno because time (and probabilities) and actions are not separated (actions are consequently not atomic). However, non-Zenoness property is required to give a meaning to models of realistic systems and is thus needed to analyze IPCs. The definition we gave to the non-Zenoness property (cf. definition 5.6) corresponds to the implementable property or bounded variability introduced in [NS91]: the number of actions performed in a given time interval is bounded. The initial definition of an implementable behavior is that it can be executed by a processor in which the measure of time is provided by a discrete clock. According to this definition, the model of hardware systems we target should be obviously implementable.

Finally, the timing model considered in IPCs is characterized by arbitrary waiting and maximal progress properties. On one hand, the arbitrary waiting property states that a process may be blocked waiting for a synchronization that is arbitrarily long (even infinitely), while still letting time advance. The arbitrary waiting ensures that no time-locks are possible. On the other hand, the maximal progress gives priority to internal action over time elapsing: an internal action cannot be delayed. Maximal progress is present in several timed process algebras [HR95, Han94, RR86] and is also used in stochastic process algebras to resolve competition between internal actions and exponentially distributed delays [Her02]. The maximal progress assumption is justified by the fact an internal action is immediately enabled and has no reason to be delayed. Note that the maximal progress is only applied to resolve concurrency between time elapsing and internal actions. Indeed, external transitions may be delayed due to the arbitrary waiting property: we have no a priori knowledge about the moment an external transition will be enabled by its environment. As for IMCs, maximal progress in IPCs is not integrated into the semantics, but is taken into account by the definition of bisimulation equivalences. For IPCs modeling a closed system, i.e., a system that does not interact with its environment anymore, we generalize maximal progress to all actions, leading to the action urgency property [NS91]. In a closed system, actions have no reason to be delayed and have always precedence over time (probabilistic) transitions of IPCs.

To summarize, compared to existing time and probabilistic process algebras in a discrete time setting, the IPC time model is mainly inspired from the time and probabilistic alternating model of Hansson [Han94], which introduces the maximal progress (called minimal delay in [Han94]) and arbitrary waiting properties. The discrete time and probabilistic process algebra defined in [And00] differs from IPCs by differentiating undelayable from delayable actions, which is a rather different explicit view of arbitrary waiting, maximal progress and action urgency properties. However, the same distinction between delayable and undelayable actions could be applied in IPCs, allowing to apply action urgency to undelayable actions during compositions. Finally, the time model of the time and probabilistic process algebra defined in [BA03] differs, because actions are not atomic but associated to a (probabilistic) delay.

8.1.2 Probabilistic Models

Two principal classes of probabilistic process algebras are distinguished: Fully probabilistic models and alternating models. In fully probabilistic models, [vGSS95] distinguishes generat-

ive, reactive or stratified models but the common property of those models is that a probability is associated to each action. Conversely, the alternating models, as in [Han94], strictly separate action-based transitions from probabilistic transitions.

The fully probabilistic models are rather different from IPCs: action and probabilities are not separated. In generative models, the next action is internally chosen according to a probabilistic distribution on the state space. This means that the sum of probabilities of outgoing transitions of each state is equal to one. If not all actions are possible, a normalization function is applied to the probabilistic distributions. In reactive models, the choice of the next action is provided by the environment, the successor state is then chosen according to a probabilistic distribution over the state space depending on the taken action. Finally, stratified models extend generative models by defining a hierarchy of probabilistic choices. Several variants of generative, reactive and stratified models exist [JLY01, DHK99, Seg95, JS90, GcJS90]. A difficulty of fully probabilistic models is the definition of parallel composition. Indeed, the interleaving semantics used in process algebra cannot be extended to probabilistic process algebra. A classical solution is the introduction of synchrony, where all processes take actions synchronously (or idle if no action is available) [GcJS90, vGSS95]. Another solution is provided by the time and probabilistic model of [BA03], which defines an asymmetric parallel composition between a generative and a reactive model.

The alternating models of [And00, Han94] are closer to IPCs: in those models probabilistic transitions are strictly separated from action-based transitions. However, there is also a strict alternation between probabilistic states, i.e., states allowing a probabilistic choice, and nondeterministic states, i.e., states allowing a nondeterministic choice. Moreover, every probabilistic transitions is followed by action transitions. In IPCs, a state can both propose a probabilistic choice (incurring also a one time step) and a nondeterministic choice. This possibility is offered by considering that probabilistic transitions are also time steps, and has the advantage to reason about the time at which an action may occur.

8.1.3 Alternating Discrete-Time and Probabilistic Models

By merging time and probabilities, IPCs combine some properties of timed process algebra and some properties of probabilistic process algebra. However, it is difficult to compare IPCs and timed process algebra abstracting from probabilistic properties. Similarly, comparing IPCs and probabilistic process algebras abstracting from time properties is difficult, as stated above.

In an alternating context, this compararison is more justified between IPCs and discrete-time and probabilistic process algebras. This limits the comparison to $\mathrm{ACP}^{+}_{\pi,\mathrm{drt}}$ [And00] and TPCCS [Han94], which also present some similarities between them. To a lesser extent, one can compare the parallel composition operator of IPCs to the one presented in [TG08] for alternating models, which, although it does not include time constructs, is mainly inspired from models obtained with TPCCS and $\mathrm{ACP}^{+}_{\pi,\mathrm{drt}}$ and presents a parallel composition operator similar to the one of IPCs.

In both $\mathrm{ACP}^{+}_{\pi,\mathrm{drt}}$ and TPCCS, time aspects and probabilistic aspects are separated, leading to three kind of transitions: probabilistic transitions, time-step transitions, and action-based transitions (time-step transitions and action-based transitions are summarized under the same name of reactive transitions). In IPCs, time and probabilistic transitions are merged limiting IPC models to two kinds of transitions: probabilistic transitions (incurring a one time-step) and action-based transitions (called interactive transitions). This difference means that IPCs are

less expressive than $\mathrm{ACP}^{+}_{\pi,\mathrm{drt}}$ and TPCCS, because IPCs cannot express a purely probabilistic functional behavior, but only probabilistic time variations (time variations can be expressed in $\mathrm{ACP}^{+}_{\pi,\mathrm{drt}}$ and TPCCS by first a probabilistic choice and then different delays). Although this functionality is important (for instance as an abstraction means of a complex arbitration policy), it is, in many cases, not essential when modeling hardware systems. Indeed, behaviors in hardware systems cannot be purely probabilistic but are deterministic.

In addition, a state of an IPC can both take interactive transitions or probabilistic transitions, which is not possible in $\mathrm{ACP}^{+}_{\pi,\mathrm{drt}}$ and TPCCS: models are alternating. We justify the possibility of taking both interactive transitions or probabilistic transitions according to the time characteristics of probabilistic transitions: either one of the interactive transitions can be taken immediately and it is taken, or no interactive transition can be taken, and time progresses with a probabilistic transition. Because it is, a priori, not possible to know if an action may be differed (due to interaction with the environment) or may do not, the two possibilities, of a nondeterministic choice between interactive transitions, or a probabilistic choice between probabilistic transitions, are kept. The situation of a state allowing to take is considered as a nondeterministic choice between a probabilistic choice and a nondeterministic choice.

$\mathrm{ACP}^{+}_{\pi,\mathrm{drt}}$ and TPCCS differ from IPC concerning nondeterministic choice between a probabilistic choice and a nondeterministic choice. In this case, the probabilistic choice is always solved first. This is justified by the fact that models in $\mathrm{ACP}^{+}_{\pi,\mathrm{drt}}$ and TPCCS has to remain alternating: a state has to be either probabilistic or reactive, but not both. For IPCs, we do not force to have an alternating model. However, for performance analysis purpose, we have to know which transition can be taken first, between an interactive one and a probabilistic one, i.e., to consider an alternating model. From IPCs, to get an alternating model as in $\mathrm{ACP}^{+}_{\pi,\mathrm{drt}}$ and TPCCS, the choice is to give priority to an interactive transition, which can be immediately taken, over a probabilistic (and timed) transition. This prioritization is called maximal progress if the interactive transition is a τ-transition, or urgency for others transitions. Maximal progress can always be applied because a τ-transition can always be taken immediately: there is no reason to delay an internal transition because it cannot interact with the environment. Urgency is postponed to closed IPCs (i.e., there are no more possible interaction with the environment) because, it is not possible, a priori, to know if an action can be taken immediately or may be delayed due to an interaction with the environment.

Concerning a nondeterministic choice between two probabilistic choices, $\mathrm{ACP}^{+}_{\pi,\mathrm{drt}}$ differs from TPCCS. In $\mathrm{ACP}^{+}_{\pi,\mathrm{drt}}$, the two probabilistic choice may be solved asynchronously, and the nondeterministic one is solved after those choices. In TPCCS, the two probabilistic choices are solved synchronously, the probabilities for this resolution being obtained by multiplying the original probabilities. IPCs follow the approach of TPCCS, because time has to pass synchronously.

$\mathrm{ACP}^{+}_{\pi,\mathrm{drt}}$ and TPCCS differ in the treatment of time progression on a probabilistic choice. In $\mathrm{ACP}^{+}_{\pi,\mathrm{drt}}$, time progression does not solve a probabilistic choice, whereas it does in TPCCS. Anew, IPCs follow clearly the view of TPCCS: since time-step transitions are also probabilistic transitions, taking a time-step transition implies that a probabilistic choice has been solved.

Concerning parallel composition, the same differences as for nondeterministic choice are encountered. In $\mathrm{ACP}^{+}_{\pi,\mathrm{drt}}$ and TPCCS, if a nondeterministic behavior is composed in parallel with a probabilistic behavior, the probabilistic choice is solved first. For IPCs, we use the classical interleaving semantics for parallelism: a probabilistic choice and a nondeterministic choice may be interleaved. In [TG08], a parallel composition is defined for general alternating

models with no time constructs (i.e., several sequential probabilistic transitions and several sequential action-based transitions are allowed). The operational semantics of the parallel composition defined in [TG08] is the same as the one for IPCs. They justify not to solve a probabilistic choice first for their parallel composition, by their capability to define a branching bisimulation equivalence that is a congruence with respect to their parallel composition operator. Actually, In [TG08], priority is given to probabilistic choices over nondeterministic choices only when considering a closed system. It is a pragmatic approach: for a closed system, they can obtain the same model as with $\mathrm{ACP}^{+}_{\pi,\mathrm{drt}}$ or TPCCS. This approach is less efficient than the one proposed for $\mathrm{ACP}^{+}_{\pi,\mathrm{drt}}$ or TPCCS because it authorizes more interleaving and consequently leads to a larger state space after composition. However, this approach as a strong justification when talking about bisimulations as we will see in the next section. Although the parallel composition operators of IPCs and of [TG08] have the same semantics, closed systems are studied with opposite methods: whilst priority is given to probabilistic choices over nondeterministic choices in [TG08], urgency gives priority to nondeterministic choices over probabilistic choices for IPCs.

A last comparison point between IPCs, $\mathrm{ACP}^{+}_{\pi,\mathrm{drt}}$, and TPCCS concerns the way probabilistic behaviors are composed in parallel. When two probabilistic behaviors are composed in parallel in $\mathrm{ACP}^{+}_{\pi,\mathrm{drt}}$, they are resolved asynchronously (using the "merge with memory" operator). In TPCCS, parallel composition between two probabilistic behaviors is defined by synchronous resolution of the probabilistic choices (as for nondeterministic choice between probabilistic behaviors). Once again, IPCs take the same point of view, justified by the time characteristics of IPC probabilistic transitions: parallel composition of probabilistic behaviors cannot be asynchronous (interleaving semantics), because time has to progress synchronously.

To summarize: we can observe that IPC formalism is closer to TPCCS formalism than to $\mathrm{ACP}^{+}_{\pi,\mathrm{drt}}$. The main difference between IPC and TPCCS concerns the forced resolution of probabilistic choices before nondeterministic ones in TPCCS, when composing different behaviors (nondeterministic choice or parallel composition). This difference is justified by the necessity of preserving an alternating model in TPCCS, while it is not needed in IPCs, thanks to the time characteristics of probabilistic transitions. The model of parallel composition of $\mathrm{ACP}^{+}_{\pi,\mathrm{drt}}$, which is claimed to be less deterministic than the one of TPCCS, cannot be applied on IPCs because probabilistic transitions of IPCs, representing also a time-step, have to be synchronous.

8.2 Bisimulations

To compare IPCs, we adapt equivalence relations called bisimulation, originally introduced for LTS. The strong bisimulation [Mil90] equates states having exactly the same functional behavior, i.e., two states s_1 and s_2 of an LTS are strongly bisimilar ($s_1 \sim s_2$) if and only if for every action a, we have:

(1) $\quad s_1 \xrightarrow{a} s_1' \Longrightarrow \left(\left(\exists s_2' \right) \quad s_2 \xrightarrow{a} s_2' \quad \wedge \quad s_2' \sim s_1' \right)$

(2) $\quad s_2 \xrightarrow{a} s_2' \Longrightarrow \left(\left(\exists s_1' \right) \quad s_1 \xrightarrow{a} s_1' \quad \wedge \quad s_1' \sim s_2' \right)$

i.e., if one state can take an action, the second one can also take the same action, and the two reached states are also strongly bisimilar. When dealing with processes with internal transition, i.e., unobservable, the strong bisimulation does not abstract from those internal steps. Among the existing weaker bisimulations in the literature, abstracting from internal trans-

itions, and thus equating more states than the strong bisimulation, the branching bisimulation is the finest equivalence [vGW96, Bas96] (i.e., it equates less states than others), but has the advantage of preserving the branching structure of processes [vGW96]. Two states s_1 and s_2 of an LTS are branching bisimilar ($s_1 \approx s_2$) if and only if for every action a, we have:

$$(1) \quad s_1 \xrightarrow{a} s_1' \Longrightarrow \left(a = \tau \wedge s_1' \approx s_2\right) \vee \left(\left(\exists s_2', s_2''\right) \quad s_2 \xrightarrow{\tau*} s_2' \xrightarrow{a} s_2'' \quad \wedge \quad s_1 \approx s_2' \wedge s_2'' \approx s_1'\right)$$

$$(2) \quad s_2 \xrightarrow{a} s_2' \Longrightarrow \left(a = \tau \wedge s_2' \approx s_1\right) \vee \left(\left(\exists s_1', s_1''\right) \quad s_1 \xrightarrow{\tau*} s_1' \xrightarrow{a} s_1'' \quad \wedge \quad s_2 \approx s_1' \wedge s_1'' \approx s_2'\right)$$

i.e., if one state can take an action, either it is a τ-transition reaching an equivalent state, or the second state can also take the same action after a (possibly empty) sequence of unobservable τ-transitions, and the two reached states remain branching bisimilar.

When constructing processes in a process algebra context, compositionally, the congruence property is required for at least the operators used for compositions. For instance, if parallel composition is congruent, the parallel composition of two bisimilar processes, with a third one, leads to two bisimilar processes.

To deal with IPC models, our goal was to define a branching bisimulation, having the ability to abstract from internal transitions, and to ensure that this bisimulation presents the congruence property, i.e., is compatible with the compositional approach we target. Because of the timed and probabilistic aspects of IPCs, related bisimulations are consequently those for timed and/or probabilistic processes.

Concerning timed processes, the branching bisimulation has been extended, for instance in [FPW05, BM02, vdZ01]. In this timed context, this bisimulation ensures that, in addition to the classical requirements for LTS, bisimilar states can take an action at the same time. The congruence property has also been taken into account [FPW08]. Nevertheless, those bisimulations are incompatible with IPCs because of the probabilistic branching of their timed transitions. However, the property of the branching bisimulation for timed processes must be preserved in the branching bisimulation for IPCs: if two states are bisimilar in an IPC, and one state can take a timed transition, the second state has to let time progress too.

Concerning probabilistic processes, the branching bisimulation has been first defined for fully probabilistic models [BH97, SL95]. The large differences between fully probabilistic models and IPCs make those bisimulations incompatible. For alternating models, which are closer to IPCs, the branching bisimulation was defined in [AW06] (only for the strictly alternating model of Hansson [Han91]). Because this branching bisimulation is not a congruence [ABDW06], a strengthened version, satisfying the congruence property, has been proposed [TG08] for general alternating models (i.e., several sequential probabilistic transitions and several sequential action-based transitions are allowed). This bisimulation is closely related to the one for IPCs, since the parallel composition operator considered in [TG08] is the same as the one of IPCs: in particular, nondeterministic choices and probabilistic choices are interleaved. Nevertheless, the bisimulation proposed in [TG08] is also incompatible with IPCs because it may equate states related by probabilistic transitions (which is justified because probabilistic transitions do not represent time progress). Thus, sequences of probabilistic steps may be amalgamated, and the requirement of bisimulation for timed processes, needed for IPCs, is broken.

Branching bisimulations defined for timed processes and probabilistic processes are consequently not applicable for IPCs. Either the conditions imposed by the time aspects are respected but probabilistic branching is not taken into account, or conditions imposed by the

probabilistic aspects are respected, but time conditions imposed by time aspects are not respected.

Actually, like IMCs aim at merging CTMCs and LTS, IPCs aim at merging DTMCs and LTS. The strong and branching bisimulations for IMCs [Her02] being based on bisimulations of LTS and of CTMCs, they inspired those for IPCs. Following the same concepts, strong and branching bisimulations of IPCs are based on bisimulations of LTS and of DTMCs (cf. section 2.2.3 of chapter 2). The possibility of equating states in a Markov chain is initially known as lumpability [KS76] and has been translated into bisimulation terms in [Her02, Buc94]. Informally, the bisimulation of probabilistic chains equates states in such a way that Markovian properties are preserved, i.e., the resulting graph is still a probabilistic chain. Bisimulations for IPCs are consequently the discrete-time analogon of the bisimulations for IMCs.

8.3 Analysis of Interactive Probabilistic Chains

A lot of toolboxes have been developped all along the years to support analysis of models obtained from process algebras. Among them, one can cite Auto/Autograph [RdS90], CWB [CPS93], Fc2Tools [BRRdS96] and Cadp [GLMS07] on which our developments are based on. System described in a process algebra are usually analyzed through an underlying model (generally a directed graph) on which analysis is processed. Depending on the process algebras (presence of time, probabilities), several underlying models exist: labeled transition systems, timed automata, Markov decision process, Semi-Markov chains, Markov chains, ...

The model underlying an IPC is a labeled transition system for functional verification, and, in general, a Markov decision process for performance evaluation. Under some conditions detailed in chapter 5 (non-Zenoness, non-determinism only due to interleaving semantics), the underlying model for performance evaluation of IPCs is a discrete-time Markov chain. In this thesis, we only tackled the performance aspects of IPCs considering discrete-time Markov chains (and not Markov decision processes).

The analysis technique of IPCs that we proposed in this thesis, relies on a performance measure, the latency, defined both on an IPC and on its underlying DTMC. Following an analytical approach based on steady state and transient state analysis of a DTMC, the latency measure may be classified in the field of DTMC performance measures based on a reward function. The latency measure is, according to us, enough generic to express a large set of performance measures on discrete-time Markov chains and thus on IPC models. The power of such an approach relies on our ability to compute the distribution of a latency, which gives a good insight of timed characteristics of a system.

Excepted from performance measures based on reward functions, We do not relate latency measure to known performance measures on DTMCs, because we are not aware of similar work concerning such a performance measure. Nevertheless, latency definition and the computation of its distribution relies on the same theoretical basis than probabilistic model-checking, i.e., measure theory [Bog07, Bil95, Coh80, KSK76]. We compute measures on different paths starting from the initial state of the discrete-time Markov chain, as in [KNP07, Seg95], and the distribution of the latency is defined on the long run, following the lines of [LHK01].

Probabilistic Model-checking could be also an interesting direction to analyze IPCs. Model-checking has emerged as a powerful tool for the automatic verification of models like LTS [QS82, CE81]. It consists in expressing a property of the specification into a temporal logic and veri-

fying automatically the model's behavior against the property. Temporal logics, initially introduced in computer science by Pnueli [Pnu77], allows to reason about behaviors of models. Model checking is recognized as a technique that allows to check "corner-cases", which are difficult to test through simulation [Kur97]. To deal with IPCs, one has to look at model checking for models including time and probabilistic extensions.

The model-checking approach has been extended to real-time systems [HNSY94, ACD93, AFH91]. In general, for real-time model checking, temporal logic formulas, dealing with time quantification (for instance the TCTL logic [ACD93]), are evaluated on timed automata [Yov96, AD94, AD91]. Several tools implement the real-time model checking approach like Uppaal [BDL+06] or Kronos [BDM+98]. Because IPCs present also probabilistic branching for timed transitions, timed temporal logics cannot be used, in general, to express properties over IPCs. However, IPCs presenting no time branching could be studied using this kind of approach.

The model-checking approach has also been extended to deal with probabilistic systems. The underlying models are, generally, Markov decision processes, discrete-time Markov chains, or continuous-time Markov chains (the latter two being a particular case of the former). Properties on these models can be expressed in logics allowing to deal with probabilities and time, for instance [BHHK03, dA97, ASB95, HJ94], and efficient tools implement the probabilistic model-checking approach, for instance Prism [KNP04].These logics could be thus applied to analyze IPCs.

Chapter 9

Conclusion

The approach proposed in this thesis tries to fulfill the wish of obtaining performance figures at a very early stage of the development of hardware systems. This wish is nowadays more and more economically justified because, for current complex architectures with their high level of parallelism, functional verification is not sufficient: compliance with the specification in terms of performance has to be ensured as well. *Early* performance study is economically justified, because the later an architecture is modified (regardless of whether this is the consequence of a functional bug or an unsatisfied performance target), the greater the cost: from few man-hours for a modification at an early stage of development, the cost may rise prohibitively (several million dollars) as the development progresses. In particular, it is currently unthinkable that, as a consequence of a design problem, several mask sets are produced for the fabrication of the same SoC. In addition to the cost of a mask set that may exceed one million dollars for the current technology (32nm), the correction of a design problem at this stage may postpone the launch of a new product by several months, which is barely acceptable in the highly competitive semiconductor market. As time to market for a new product gets shorter, one cannot afford deep modifications of a design at an advanced stage of its development.

Up to now, no industrial methodology is available for precise performance studies of preliminary hardware specifications. Widespread methods rely on simulations, for which performance results are highly dependent on both the simulation engine and on the modeling formalism. Furthermore, models used for simulation often rely on informal assumptions that, despite their obvious correctness, may turn out to be inaccurate. Finally, due to increasing complexity of current designs, simulation is less and less capable of producing, in an acceptably short time interval, performance figures with a reasonable confidence interval.

This thesis proposes a methodology for performance studies of hardware models, which is a step towards providing designers with measures guiding them in their architectural choices. The industrial context of this thesis imposed several initial requirements on the design of the method.

A first requirement was the use of formal methods to ensure a certain confidence in the resulting performance measures. A second requirement was the scalability of the approach, because the complexity of the targeted systems is due to their high level of parallelism. Furthermore, the approach had to support any kind of hardware systems. In particular, the performance analysis had to be generic, in the sense that performance measures give, whatever the particular studied system, a significant insight of its behavior. Finally, the methodology

had to remain accessible to non-experts if to be spread in industry.

Initially, the formalism of IMCs seemed a very elegant and adapted answer to all those requirements. Firstly, IMCs enable the generation of CTMCs in a hierarchical and compositional way, which circumvents the usual state space explosion problem and provides the researched scalability. Secondly, the underlying model for IMCs, CTMCs, is simple and adequate and can be used to estimate a wide range of performance characteristics. In addition, IMCs remain compatible with purely functional models, on which functional verification can be performed. Using the very same model for functional verification and performance evaluation reduces the overall cost and improves the confidence in the results. Finally, by using continuous phase-type distributions, IMCs allow to model any kind of systems.

Unfortunately, using a continuous-time setting, such as IMCs, to model a predominantly discrete-time system, such as hardware architectures, requires approximation of discrete delays by continuous phase-type distributions. As illustrated in chapter 4, we were not able to evaluate the impact of these approximations in terms of error on the computed performance results. Consequently, we could not convince the architects of STMicroelectronics to trust results given by the IMC approach.

The approach introduced in this thesis solves this issue by transposing IMCs in a discrete time setting, leading to the concept of IPCs. Like IMCs, IPCs enable the generation of Markov chains in a hierarchical and compositional way. By using discrete phase-type distributions, one can model any kind of discrete-time system.

The contributions of this thesis actually consist of two separated, but complementary results: the IPC modeling approach and the definition of the latency performance measure. Those two parts are linked by a transformation from an IPC to a DTMC on which one can compute latencies.

Concerning the IPC modeling, the transposition from the continuous-time context of IMCs to the discrete-time context of IPCs is not immediate, mainly because it implies a different model of elapsing time: whilst the memoryless exponential distributions used in IMCs allow to interleave delays in parallel, discrete-time delays elapsing in parallel in IPCs have to progress synchronously. However, our definition of IPCs keeps the strong points of IMCs. Firstly, we separate delays from actions which simplifies the definition of synchronization. The construction of IPCs relies on the definition of a language, $\mathbb{IPC}_L$, with a semantics authorizing a hierarchical and compositional approach. Additionally, we define bisimulations operating on IPCs, as useful means to aggregate IPCs and to circumvent the state space explosion problem. IPC models also support *maximal progress* cut without preventing the use of bisimulations: internal actions are taken without letting time elapse. For the characterization of time in IPCs, we use the so-called *arbitrary waiting* property, which states that a functionally blocked process cannot prevent other processes to evolve in time (which avoids time locks). This first part of the work enables to model hardware systems accurately.

To transform an IPC into a DTMC, we apply the *urgency* assumption, which imposes that no action can let time elapse. This assumption is only authorized if the system is closed, i.e., if it cannot interact with its environment anymore. In other words, the system must not be under-specified. The proposed transformation turns a closed IPC into a DTMC annotated with functional information.

On this DTMC, we can compute a generic performance measure, the latency. Besides defining latency formally, we provide an algorithmic definition of their long-run average distri-

bution, which only depends on state-probabilities of the DTMC and extracted sub-DTMCs. In our opinion, the latency is a generic measure that covers a wide range of specific measures (covering both latency and throughput).

Both parts of this work (modeling hardware systems and performance study based on latency) resulted in the development of software tools based on the efficient APIs available in the CADP toolbox. Those tools have been applied to an industrial case-study: the xSTREAM architecture. This confirmed that our proposed approach scales up to large systems and, in practice, the use of the latency measure enables to evaluate hardware design choices.

Future Work

Although this work led to interesting results, we are still far from a complete industrial methodology, answering the problem of performance evaluation of hardware designs. We identified several directions for further research, covering both the modeling approach and the performance studies.

Modeling approach

A first research direction can focus on the practical limitation of the use of a process algebra (in LOTOS-style for IPCs) to model hardware systems, especially in an industrial context: engineers are not used to this kind of language, which is an obstacle to a fast assimilation of the modeling methodology. This problem can be tackled by the use of more recent languages such as LOTOS NT [GS98], which propose programming paradigms closer to the ones of well-known languages (like loops, etc.). Nevertheless, using a new modeling language different from those used in the established design flows of companies, results in a redundant effort. A translation from the models of the existing design flows to process algebra would certainly accelerate the spread of formal methods in the industry. Some promising results in this direction have been already obtained, for instance the approach consisting to translate SYSTEMC/TLM models (transaction-level modeling) to LOTOS models [GHPS09].

A second research direction concerns the improvement of the expressiveness of IPCs, by allowing to take into account purely probabilistic behaviors. Indeed, at the present time, a probabilistic decision is only possible when time progresses, as delays are modelled using probabilistic distributions. One could also want to model probabilistic systems where a probabilistic choice is processed instantaneously. Moreover, an instantaneous probabilistic choice could be an elegant solution to model complex arbitration policies in an abstract and concise way. This possibility could be enabled by taking into account the work presented for (untimed) alternating models in [TG08]. Their parallel composition operator being similar to the one of this thesis, the two approaches could be merged to deal with both purely probabilistic behaviors and time probabilistic behaviors. Consequently, we would have to distinguish two types of probabilistic transitions, the ones taking one time step, i.e., the timed probabilistic transitions, and the others that are fired instantaneously, i.e., the untimed probabilistic transitions. At first sight, we should treat separately parallel composition of untimed probabilistic transitions and parallel composition of timed probabilistic transitions. On the composed model, we would give priority to interactive transitions over timed probabilistic transitions (urgency cut) and we would give priority to instantaneous probabilistic transitions over interactive transitions,

as in [TG08] (and thus, untimed probabilistic transitions have priority on timed probabilistic transitions in our case). This merge would certainly lead to a new branching bisimulation relation covering both the one of [TG08] and ours.

A third research direction can be to study how discrete-time and continuous-time approaches could be simultaneously taken into account. So far, they are strictly separated in the CADP toolbox but, for instance, it would be interesting to focus on how IPC and IMC approaches could be merged in a common formalism. In chapter 4, we only briefly investigated the relation between the semantics of IPC and IMC. A comparative study of both IPC and IMC formalism could also provide some answers concerning errors due to delay approximations in the models. One could for instance study the error on the steady-state probabilities when approximating an IPC by an IMC and vice-versa. Simulation is also one of the possible analysis technique when considering a system presenting both probabilistic (timed) transitions and Markovian transitions.

Performance Evaluation

Concerning performance evaluation, a first research direction concerns the possibility to use existing performance methodologies with IPCs. The latency distribution we defined is, we believe, an easily computable performance measure that provides interesting insights of the timed behaviors of studied systems. However, other interesting performance measures could be obtained, for instance by adapting model-checking techniques to IPCs.

A second research direction regards the possibility of adapting the latency definition to continuous time Markov chains, which would allow to get similar performance results using IMCs instead of IPCs. This implies to study the adaptability of the latency measure to CTMCs, and the possible transformation of IMCs into annotated CTMCs.

Finally, a third research direction can be to limit, as much as possible, the size of the generated DTMCs. To do so, the approach developed throughout this thesis (complete exploration of the model state space) does not seem to be the most adequate. Indeed, generated DTMCs may be huge, even if minimization has been applied at intermediate steps of the compositional construction. Another solution would be to not generate the DTMC exhaustively, but to keep the system as a set of communicating components and try to obtain the same performance measures by local study of components. In this aim, we could explore relations between our approach and the stochastic automata networks [Pla85], which enable to describe multidimensional Markov chains, avoiding to store their complete state space.

Appendix A

Proof of Branching Equivalence Between Urgency-Cut IPCs of Branching Equivalent IPCs

In this appendix, we prove the lemma 5.5, i.e., if two IPCs are branching bisimilar, the two corresponding urgency-cut IPCs remain branching bisimilar.

Lemma A.1. *The two urgency cut IPCs of two branching bisimilar IPCs are branching bisimilar:*

$$\left(\forall (P_1, P_2) \in \mathscr{P}\right) \quad (P_1 \approx P_2) \Longrightarrow \left(P_{1\not\rightarrow_{\mathcal{A}}} \approx P_{2\not\rightarrow_{\mathcal{A}}}\right)$$

Proof. Consider two IPCs $P_1 = \langle S_1, \mathcal{A}, \longrightarrow, \rightsquigarrow, s_1\rangle$ and $P_2 = \langle S_2, \mathcal{A}, \longrightarrow, \rightsquigarrow, s_2\rangle$ such that $P \approx P_1$. We want to prove that the two urgency cut IPCs $P_{1\not\rightarrow_{\mathcal{A}}} = \langle S_1', \mathcal{A}, \longrightarrow, \rightsquigarrow, s_1\rangle$ and $P_{2\not\rightarrow_{\mathcal{A}}} = \langle S_2', \mathcal{A}, \longrightarrow, \rightsquigarrow, s_2\rangle$ are also branching bisimilar.

By definition, we have $S_1' \subseteq S_1$ and $S_2' \subseteq S_2$. We consider the branching bisimulation $\approx$ over the states of $S_1 \cup S_2$, i.e., $s_i \approx s_j$ means that s_i and s_j are branching bisimilar in P_1 and P_2, and we define the relation $\mathcal{R}$ over the states of $S_1' \cup S_2'$, i.e., $\mathcal{R}$ is defined only for states of the urgency-cut IPCs:

$$\mathcal{R} = \left\{\left(s_i, s_j\right) \mid s_i, s_j \in S_1' \cup S_2' \text{ and } s_i \approx s_j\right\}$$

We show that $\mathcal{R}$ is a branching bisimulation. Clearly, $\mathcal{R}$ is an equivalence relation because $\approx$ is an equivalence relation.

Suppose $s_1 \ \mathcal{R} \ s_2$ and consider an equivalence class $\mathcal{C}_{\mathcal{R}}$ induced by $\mathcal{R}$ in $P_{1\not\rightarrow_{\mathcal{A}}}$. Because $s_1 \ \mathcal{R} \ s_2$, we have also $s_1 \approx s_2$.

Firstly, suppose that there is an action a and a state $s_1' \in \mathcal{C}_{\mathcal{R}}$, such that $s_1 \xrightarrow{a} s_1'$. This implies that $\gamma_0(s_1, a, \left[s_1'\right]_{/_{\mathcal{R}}}) = \mathsf{True}$ and $\gamma_0(s_1, a, \left[s_1'\right]_{/_{\approx}}) = \mathsf{True}$. From $s_1 \approx s_2$ we have:

- either $a = \tau$ and $s_2 \in \left[s_1'\right]_{/_{\approx}}$ and we can deduce that $s_2 \in \left[s_1'\right]_{/_{\mathcal{R}}}$.
- or there is a state $s_2'' \not\xrightarrow{\tau}$ such that $s_2 \xrightarrow{\tau*} s_2''$ and $s_2'' \approx s_2$ satisfying $\gamma_0(s_2'', a, \left[s_1'\right]_{/_{\approx}}) = \mathsf{True}$. Thus there is a state $s_2' \in \left[s_1'\right]_{/_{\approx}}$ such that $s_2'' \xrightarrow{a} s_2'$. From $s_2' \approx s_1'$, and because s_2' is reachable from s_2 by taking interactive transitions (which are never cut by urgency), we deduce $s_2' \ \mathcal{R} \ s_1'$.

We have found a state $s_2'' \stackrel{\tau}{\not\longrightarrow}$, such that $s_2 \stackrel{\tau*}{\longrightarrow} s_2''$ and $s_2'' \mathcal{R} s_2$ (because $s_2'' \approx s_2$), that satisfies $\gamma_0(s_1, a, \left[s_1'\right]_{/\mathcal{R}}) = \gamma_0(s_2'', a, \left[s_1'\right]_{/\mathcal{R}}) = \text{True}$.

Secondly, suppose that there is a probability $p \in]0,1]$ and a state $s_1' \in \mathcal{C}_{\mathcal{R}}$, such that $s_1 \stackrel{p}{\rightsquigarrow} s_1'$. Because $s_1' \in \mathcal{C}_{\mathcal{R}}$, we have $s_1' \in S_1'$ and we can deduce that $\left(\nexists a \in \mathcal{A}\right) \quad s_1 \stackrel{a}{\longrightarrow}$ (otherwise, the transition $s_1 \stackrel{p}{\rightsquigarrow} s_1'$ would have been cut by urgency).

From $s_1 \approx s_2$, and $s_1 \stackrel{\tau}{\not\longrightarrow}$, we can deduce that there is a state $s_2'' \stackrel{\tau}{\not\longrightarrow}$, such that $s_2'' \approx s_2$, $s_2 \stackrel{\tau*}{\longrightarrow} s_2''$ and satisfying $\gamma_P(s_1, \left[s_1'\right]_{/\approx}) = \gamma_P(s_2'', \left[s_1'\right]_{/\approx})$.

We can commpute γ_P from s_1 to the class $\mathcal{C}_{\mathcal{R}}$:

$$\gamma_P(s_1, \mathcal{C}_{\mathcal{R}}) \quad = \quad \gamma_P\left(s_1, [s_1']_{/\mathcal{R}}\right)$$

By definition of $\mathcal{R}$, we have $\gamma_P\left(s_1, \left[s_1'\right]_{/\mathcal{R}}\right) \leq \gamma_P\left(s_1, \left[s_1'\right]_{/\approx}\right)$. Actually, we want to prove that

$$\gamma_P\left(s_1, [s_1']_{/\mathcal{R}}\right) = \gamma_P\left(s_1, [s_1']_{/\approx}\right)$$

To that purpose, suppose that $\gamma_P\left(s_1, \left[s_1'\right]_{/\mathcal{R}}\right) < \gamma_P\left(s_1, \left[s_1'\right]_{/\approx}\right)$. This means that there is a state $\tilde{s}_1 \in S \backslash S_1$ such that $\tilde{s}_1 \in \left[s_1'\right]_{/\approx}$ and $s_1 \rightsquigarrow \tilde{s}_1$ in P_1. In other words $\tilde{s}_1$ is a state of P_1 and not of $P_{1\not\rightsquigarrow_{\mathcal{A}}}$ in the equivalence class of s_1' for $\approx$. The transition $s_1 \rightsquigarrow \tilde{s}_1$ has consequently been cut by the urgency. Thus, $\left(\exists a \in \mathcal{A}\right)$, such that $s_1 \stackrel{a}{\longrightarrow}$ in P_1, that is in contradiction with $\left(\nexists a \in \mathcal{A}\right) \quad s_1 \stackrel{a}{\longrightarrow}$. We can conclude that

$$\gamma_P\left(s_1, [s_1']_{/\mathcal{R}}\right) = \gamma_P\left(s_1, [s_1']_{/\approx}\right)$$

As a consequence, we have:

$$\gamma_P(s_1, \mathcal{C}_{\mathcal{R}}) \quad = \quad \gamma_P\left(s_1, [s_1']_{/\mathcal{R}}\right) \quad = \quad \gamma_P\left(s_1, [s_1']_{/\approx}\right) \quad = \quad \gamma_P\left(s_2'', [s_1']_{/\approx}\right)$$

Because $s_2'' \approx s_1$ and there are no interactive transitions from s_1, we can deduce that there are no interactive transitions from s_2''. Consequently, there is no probabilistic transition from s_2'' cut by urgency. Thus, $\gamma_P\left(s_2'', \left[s_1'\right]_{/\approx}\right) = \gamma_P\left(s_2'', \left[s_1'\right]_{/\mathcal{R}}\right)$, and we can conclude that:

$$\gamma_P(s_1, \mathcal{C}_{\mathcal{R}}) \quad = \quad \gamma_P\left(s_2'', [s_1']_{/\approx}\right) \quad = \quad \gamma_P\left(s_2'', [s_1']_{/\mathcal{R}}\right) \quad = \quad \gamma_P(s_2'', \mathcal{C}_{\mathcal{R}}) \quad =$$

We have found a state $s_2'' \stackrel{\tau}{\not\longrightarrow}$, such that $s_2 \stackrel{\tau*}{\longrightarrow} s_2''$ and $s_2'' \mathcal{R} s_2$ (because $s_2'' \approx s_2$), that satisfies $\gamma_P(s_1, \left[s_1'\right]_{/\mathcal{R}} = \gamma_P(s_2'', \left[s_1'\right]_{/\mathcal{R}}) = \text{True}$.

We can conclude that $\mathcal{R}$ is a branching probabilistic bisimulation. □

Appendix B

Proof of the Congruence Theorem

In this appendix, we prove theorem 5.1, i.e., we prove that IPC probabilistic branching bisimulation is a congruence with respect to the parallel composition operator.

For convenience, B denotes both the behavior $B \in \mathcal{B}$ and the initial state of the IPC on which B can be mapped on; this convention allows to use both notations defined for behaviors of $\mathsf{IPC}_{\mathsf{L}}$ and notations defined for states of an IPC,

First of all, we introduce several preliminary lemmas.

Lemma B.1. *For all $B_1, B_2, B_1', B_2' \in \mathcal{B}$ and $A \subseteq \mathcal{A}\backslash\{\tau\}$, we have:*

$$B_1 \xrightarrow{\tau^*} B_1' \wedge B_2 \xrightarrow{\tau^*} B_2' \implies B_1 \,|[A]|\, B_2 \xrightarrow{\tau^*} B_1' \,|[A]|\, B_2'$$

Proof. If $B_1 \xrightarrow{\tau} \widetilde{B}_1$, we have that $B_1 |[A]| B_2 \xrightarrow{\tau} \widetilde{B}_1 |[A]| B_2$. Using this property that is directly deduced from semantic rules, the lemma can be proved by induction on the lengths of the paths defined by $B_1 \xrightarrow{\tau^*} B_1'$ and $B_2 \xrightarrow{\tau^*} B_2'$. □

Lemma B.2. *For all $B_1, B_2, B_1', B_2' \in \mathcal{B}$ and $A \subseteq \mathcal{A}\backslash\{\tau\}$, we have:*

$$\gamma_P(B_1 \,|[A]|\, B_2, \{B_1' \,|[A]|\, B_2'\}) = \gamma_P(B_1, \{B_1'\}) \times \gamma_P(B_2, \{B_2'\})$$

Proof. Suppose that $\gamma_P\left(B_1, \left\{B_1'\right\}\right) = 0$ or $\gamma_P\left(B_2, \left\{B_2'\right\}\right) = 0$, i.e., there are no probabilistic transitions between B_1 and B_1' or between B_2 and B_2'. According to semantic rules, we can deduce that there are no probabilistic transitions between $B_1|[A]|B_2$ and $B_1'|[A]|B_2'$, thus $\gamma_P\left(B_1 \,|[A]|\, B_2, \left\{B_1' \,|[A]|\, B_2'\right\}\right) = 0$.

Now, suppose that $\gamma_P\left(B_1, \left\{B_1'\right\}\right) > 0$ and $\gamma_P\left(B_2, \left\{B_2'\right\}\right) > 0$. We can deduce that $\left(\exists p_1, \ldots, p_i, q_1, \ldots, q_j \in]0,1]\right)$ such that $B_1 \overset{p_k}{\rightsquigarrow} B_1'$ for all $k \leq i$, $B_2 \overset{p_l}{\rightsquigarrow} B_2'$ for all $l \leq j$, $\gamma_P\left(B_1, \left\{B_1'\right\}\right) = \sum_{k \leq i} p_k$, and $\gamma_P\left(B_2, \left\{B_2'\right\}\right) = \sum_{l \leq j} p_l$.

According to semantic rules, we can say that $B_1 |[A]| B_2 \overset{p_k\, p_l}{\rightsquigarrow} B_1' |[A]| B_2'$ for all $k \leq i$ and $l \leq j$.

It follows that:

$$\begin{aligned}\gamma_P(B_1 \,|[A]|\, B_2, \{B_1' \,|[A]|\, B_2'\}) &= \sum_{k\le i \wedge l \le j} p_k\, p_l = \sum_{k\le i}\sum_{l\le j} p_k\, p_l \\ &= \left(\sum_{k\le i} p_k\right)\left(\sum_{l\le j} p_l\right) = \gamma_P(B_1, \{B_1'\}) \times \gamma_P(B_2, \{B_2'\})\end{aligned}$$

□

We can now prove the congruence property of the probabilistic branching bisimulation defined in section 5.8

Theorem B.1. *Probabilistic branching bisimulation is a congruence with respect to the parallel composition operator.*

Proof. Let $\mathcal{R} = \{(B_1 \,|[A]|\, B_3, B_2 \,|[A]|\, B_4) \mid B_1, B_2, B_3, B_4 \in \mathcal{B},\ B_1 \approx B_2,\ B_3 \approx B_4\}$.

In the following, when we write "$B_i' \,|[A]|\, B_j'\ \mathcal{R}\ B_i \,|[A]|\, B_j$", we assume implicitly that $B_i' \approx B_i$ and $B_j' \approx B_j$.

We show that $\mathcal{R}$ is a branching bisimulation relation. First of all, we can say that it is an equivalence:

- Reflexivity: $B_1 \,|[A]|\, B_2\ \mathcal{R}\ B_1 \,|[A]|\, B_2$ (because $\approx$ is also reflexive: $B_1 \approx B_1$ and $B_2 \approx B_2$)
- Symmetry: if $B_1 \,|[A]|\, B_2\ \mathcal{R}\ B_3 \,|[A]|\, B_4$, then $B_3 \,|[A]|\, B_4\ \mathcal{R}\ B_1 \,|[A]|\, B_2$ (because $\approx$ is also symmetric: $B_1 \approx B_3 \Longleftrightarrow B_3 \approx B_1$ and $B_2 \approx B_4 \Longleftrightarrow B_4 \approx B_2$)
- Transitivity : if $B_1|[A]|B_2\ \mathcal{R}\ B_3|[A]|B_4$ and $B_3|[A]|B_4\ \mathcal{R}\ B_5|[A]|B_6$, then $B_1|[A]|B_2\ \mathcal{R}\ B_5|[A]|B_6$ (because $\approx$ is also transitive: $B_1 \approx B_3 \wedge B_3 \approx B_5 \Longrightarrow B_1 \approx B_5$ and $B_2 \approx B_4 \wedge B_4 \approx B_6 \Longrightarrow B_2 \approx B_6$)

Let $B_1, B_2, B_3, B_4 \in \mathcal{B}$ be four states such that $B_1 \,|[A]|\, B_3\ \mathcal{R}\ B_2 \,|[A]|\, B_4$ for some $A \subseteq \mathcal{A}\backslash\{\tau\}$.

(i) Suppose that $B_1 \,|[A]|\, B_3 \xrightarrow{a}$.

- If $a \notin A$, we distinguish two cases: $\left(\exists B_1'\right)\ B_1 \xrightarrow{a} B_1'$ or $\left(\exists B_3'\right)\ B_3 \xrightarrow{a} B_3'$. Because a similar reasoning can be followed for the two cases, without loss of generality, we only treat the case $\left(\exists B_1'\right)\ B_1 \xrightarrow{a} B_1'$.
  From $B_1 \approx B_2$, we can deduce that
  - Either $a = \tau$ and $B_2 \in \left[B_1'\right]_{/\approx}$, i.e., $B_2 \approx B_1'$. Then $B_2 \,|[A]|\, B_3\ \mathcal{R}\ B_1' \,|[A]|\, B_3$.
    Under the hypothesis $\gamma_0\left(B_1 \,|[A]|\, B_3, \tau, \left[B_1' \,|[A]|\, B_3\right]_{/\mathcal{R}}\right) = \mathsf{True}$, the state $B_2 \,|[A]|\, B_3$ satisfies $B_2 \,|[A]|\, B_3 \in \left[B_1' \,|[A]|\, B_3\right]_{/\mathcal{R}}$.
  - Or $\gamma_0\left(B_1, a, \left[B_1'\right]_{/\approx}\right) = \gamma_0\left(B_2', a, \left[B_1'\right]_{/\approx}\right)$ for some B_2' such that $B_2 \xrightarrow{\tau*} B_2'$ and $B_2' \approx B_1$.
    Consequently, there exists $B_2'' \in \left[B_1'\right]_{/\approx}$ such that $B_2' \xrightarrow{a} B_2''$. If B_2' can reach B_2'' with a, we deduce that $B_2' \,|[A]|\, B_3 \xrightarrow{a} B_2'' \,|[A]|\, B_3$, i.e., $\gamma_0\left(B_2' \,|[A]|\, B_3, a, \left[B_1' \,|[A]|\, B_3\right]_{/\mathcal{R}}\right) = \mathsf{True}$.
    Under the hypothesis $\gamma_0\left(B_1 \,|[A]|\, B_3, a, \left[B_1' \,|[A]|\, B_3\right]_{/\mathcal{R}}\right) = \mathsf{True}$, we have found the state $B_2' \,|[A]|\, B_3$ that satisfies $\gamma_0\left(B_2' \,|[A]|\, B_3, a, \left[B_1' \,|[A]|\, B_3\right]_{/\mathcal{R}}\right) = \mathsf{True}$, $B_2 \,|[A]|\, B_3 \xrightarrow{\tau*} B_2' \,|[A]|\, B_3$ (c.f. lemma B.1), and $B_2' \,|[A]|\, B_3\ \mathcal{R}\ B_1 \,|[A]|\, B_3$.

– If $a \in A$ ($a \neq \tau$), we know that B_1 and B_3 have to take a synchronously, i.e., there exists B_1' and B_3' such that $B_1 \xrightarrow{a} B_1'$ and $B_3 \xrightarrow{a} B_3'$. Because $B_1 \approx B_2$ and $B_3 \approx B_4$, we can deduce that
  – $\gamma_0\left(B_1, a, \left[B_1'\right]_{/_\approx}\right) = \gamma_0\left(B_2', a, \left[B_1'\right]_{/_\approx}\right)$ for some B_2' such that $B_2 \xrightarrow{\tau*} B_2'$ and $B_2' \approx B_1$.
  – $\gamma_0\left(B_3, a, \left[B_3'\right]_{/_\approx}\right) = \gamma_0\left(B_4', a, \left[B_3'\right]_{/_\approx}\right)$ for some B_4' such that $B_4 \xrightarrow{\tau*} B_4'$ and $B_4' \approx B_3$.

  Consequently, there are $B_2'' \in \left[B_1'\right]_{/_\approx}$ such that $B_2' \xrightarrow{a} B_2''$ and $B_4'' \in \left[B_3'\right]_{/_\approx}$ such that $B_4' \xrightarrow{a} B_4''$. This implies that $B_2' |[A]| B_4' \xrightarrow{a} B_2'' |[A]| B_4''$, i.e $\gamma_0\left(B_2' |[A]| B_4', a, \left[B_1' |[A]| B_3'\right]_{/_\mathcal{R}}\right) =$ True.

  Under the hypothesis $\gamma_0\left(B_1 |[A]| B_3, a, \left[B_1' |[A]| B_3'\right]_{/_\mathcal{R}}\right) =$ True, we have the state $B_2' |[A]| B_4'$ that satisfies $\gamma_0\left(B_2' |[A]| B_4', a, \left[B_1' |[A]| B_3'\right]_{/_\mathcal{R}}\right) =$ True, $B_2 |[A]| B_4 \xrightarrow{\tau*} B_2' |[A]| B_4'$ (c.f. lemma B.1), and $B_2' |[A]| B_4' \;\mathcal{R}\; B_1 |[A]| B_3$.

(ii) Suppose that $B_1 |[A]| B_3 \xrightarrow{\tau} \!\!\!\!\!/\;\;$ and consider an equivalence class $\mathcal{C} = \left[B_i |[A]| B_j\right]_{/_\mathcal{R}}$ for some states $B_i, B_j \in \mathcal{B}$.

We can directly compute $\gamma_P(B_1 |[A]| B_3, \mathcal{C})$:

$$
\begin{aligned}
\gamma_P(B_1 |[A]| B_3, \mathcal{C}) &= \sum_{B_i' |[A]| B_j' \in \mathcal{C}} \gamma_P\left(B_1 |[A]| B_3, \left\{B_i' |[A]| B_j'\right\}\right) \\
&= \sum_{B_i' |[A]| B_j' \in \mathcal{C}} \gamma_P\left(B_1, \left\{B_i'\right\}\right) \times \gamma_P\left(B_3, \left\{B_j'\right\}\right) \qquad \text{(cf. lemma B.2)} \\
&= \sum_{B_i' \in [B_i]_{/_\approx} \wedge B_j' \in \left[B_j\right]_{/_\approx}} \gamma_P\left(B_1, \left\{B_i'\right\}\right) \times \gamma_P\left(B_3, \left\{B_j'\right\}\right) \\
&= \sum_{B_i' \in [B_i]_{/_\approx}} \sum_{B_j' \in \left[B_j\right]_{/_\approx}} \gamma_P\left(B_1, \left\{B_i'\right\}\right) \times \gamma_P\left(B_3, \left\{B_j'\right\}\right) \\
&= \left(\sum_{B_i' \in [B_i]_{/_\approx}} \gamma_P\left(B_1, \left\{B_i'\right\}\right)\right) \times \left(\sum_{B_j' \in \left[B_j\right]_{/_\approx}} \gamma_P\left(B_3, \left\{B_j'\right\}\right)\right) \\
&= \gamma_P\left(B_1, [B_i]_{/_\approx}\right) \times \gamma_P\left(B_3, \left[B_j\right]_{/_\approx}\right)
\end{aligned}
$$

Because $B_1 |[A]| B_3 \xrightarrow{\tau} \!\!\!\!\!/\;\;$, we can deduce that $B_1 \xrightarrow{\tau} \!\!\!\!\!/\;\;$ and $B_3 \xrightarrow{\tau} \!\!\!\!\!/\;\;$. From $B_2 \approx B_1$ and $B_4 \approx B_3$, we can also find $B_2' \xrightarrow{\tau} \!\!\!\!\!/\;\;$ and $B_4' \xrightarrow{\tau} \!\!\!\!\!/\;\;$ satisfying:
– $\gamma_P\left(B_1, [B_i]_{/_\approx}\right) = \gamma_P\left(B_2', [B_i]_{/_\approx}\right)$, with $B_2 \xrightarrow{\tau*} B_2'$ and $B_2' \approx B_1$.
– $\gamma_P\left(B_3, \left[B_j\right]_{/_\approx}\right) = \gamma_P\left(B_4', \left[B_j\right]_{/_\approx}\right)$, with $B_4 \xrightarrow{\tau*} B_4'$ and $B_4' \approx B_3$.

And we can write:

$$\begin{aligned}
\gamma_P(B_1 \,|[A]|\, B_3, \mathcal{C}) &= \gamma_P\left(B_1, [B_i]_{/\approx}\right) \times \gamma_P\left(B_3, \left[B_j\right]_{/\approx}\right) = \gamma_P\left(B_2', [B_i]_{/\approx}\right) \times \gamma_P\left(B_4', \left[B_j\right]_{/\approx}\right) \\
&= \left(\sum_{B_i' \in [B_i]_{/\approx}} \gamma_P\left(B_2', \left\{B_i'\right\}\right)\right) \times \left(\sum_{B_j' \in \left[B_j\right]_{/\approx}} \gamma_P\left(B_4', \left\{B_j'\right\}\right)\right) \\
&= \sum_{B_i' \in [B_i]_{/\approx}} \sum_{B_j' \in \left[B_j\right]_{/\approx}} \gamma_P\left(B_2', \left\{B_i'\right\}\right) \times \gamma_P\left(B_4', \left\{B_j'\right\}\right) \\
&= \sum_{B_i' \in [B_i]_{/\approx} \wedge B_j' \in \left[B_j\right]_{/\approx}} \gamma_P\left(B_2', \left\{B_i'\right\}\right) \times \gamma_P\left(B_4', \left\{B_j'\right\}\right) \\
&= \sum_{B_i' |[A]| B_j' \in \mathcal{C}} \gamma_P\left(B_2', \left\{B_i'\right\}\right) \times \gamma_P\left(B_4', \left\{B_j'\right\}\right) \\
&= \sum_{B_i' |[A]| B_j' \in \mathcal{C}} \gamma_P\left(B_2' \,|[A]|\, B_4', \left\{B_i' \,|[A]|\, B_j'\right\}\right) \qquad \text{(cf. lemma B.2)} \\
&= \gamma_P\left(B_2' \,|[A]|\, B_4', \mathcal{C}\right)
\end{aligned}$$

Under the hypothesis $B_1 \,|[A]|\, B_3 \stackrel{\tau}{\not\longrightarrow}$, we have found a state $B_2' \,|[A]|\, B_4' \stackrel{\tau}{\not\longrightarrow}$ that satisfies $\gamma_M(B_1 \,|[A]|\, B_3, \mathcal{C}) = \gamma_M\left(B_2' \,|[A]|\, B_4', \mathcal{C}\right)$, $B_2 \,|[A]|\, B_4 \stackrel{\tau^*}{\longrightarrow} B_2' \,|[A]|\, B_4'$ (cf. lemma B.1), and $B_2' \,|[A]|\, B_4' \;\mathcal{R}\; B_1 \,|[A]|\, B_3$.

□

Appendix C

Proof of Strong Equivalence Between DTAMCs Associated to Branching Bisimilar Time Deterministic IPCs

In this appendix, we prove the lemma 5.8, i.e., that the DTAMCs associated to two branching bisimilar urgency-cut tdIPCs are strongly bisimilar. In a first section, we show that two branching bisimilar urgency-cut tdIPCs can be scheduled to get two branching bisimilar dIPCs. In a second section, we prove that the DTAMCs associated to branching bisimilar dIPCs are strongly bisimilar. Finally, we summarize to conclude on the strong bisimulation equivalence of DTAMCs associated to branching bisimilar urgency-cut tdIPCs.

C.1 Scheduling of Bisimilar Urgency-Cut Time Deterministic Interactive Probabilistic Chains

In this section, we want to prove that two branching bisimilar urgency-cut tdIPCs can be scheduled in such a way that two branching bisimilar dIPCs are obtained. To that purpose, we first define the notions of path in an IPC, trace and scheduler. Because there is no possible misunderstanding, we overload notations concerning paths in a DTMC (cf. chapter 3), for paths in an IPC. In addition, we need a definition of scheduler on IPCs that is more general than the one presented in chapter 5. Consequently, we redefine the notion of scheduler that will be used throughout this appendix.

Definition C.1 (Finite path in an IPC)**.** *Given an IPC* $P = \langle S, \mathcal{A}, \longrightarrow, \rightsquigarrow, s_0 \rangle$*, a finite path* σ *is an alternating sequence* $\sigma = s_1\ l_1\ s_2\ l_2\ s_2\ \ldots\ l_{n-1}\ s_n$*, where for every* $i \leq n$*,* $s_i \in S$*,* $l_i \in \mathcal{A} \cup]0,1]$ *and* $s_i \xrightarrow{l_i} s_{i+1}$ *or* $s_i \overset{l_i}{\rightsquigarrow} s_{i+1}$*.*

Let $last(\sigma)$ denotes the last state s_n of the finite path "$\sigma = s_1\ l_1\ s_2\ l_2\ s_2\ \ldots\ l_{n-1}\ s_n$". Given a state $s \in S$, $Paths(s)$ denotes the set of finite paths starting in s. We define the concatenation operator $\circ$ over paths: $\forall \sigma, \sigma', \sigma \circ \sigma'$ is defined only if $\sigma' \in Paths(last(c))$.

For convenience, we authorize writing paths using the transition notations, for instance

$$\sigma = s_0 \xrightarrow{\tau} s_1 \overset{p}{\rightsquigarrow} s_2 \xrightarrow{a} \xrightarrow{\tau*} s_3$$

where intermediate states may be omitted, if not needed.

Using definition C.1, we can define the notion of trace. Informally, the trace of a path represents the moves done in the path. Probabilistic moves are represented by the symbol $\mathfrak{P}$, but concrete probabilities are not mentioned.

Definition C.2 (Concrete trace)**.** *Given an IPC $P = \langle S, \mathcal{A}, \longrightarrow, \rightsquigarrow, s_0\rangle$, the concrete trace of a path σ is a sequence $\mathsf{tr}(\sigma) = t_1\ t_2\ \ldots\ t_m$ where for every $i \leq m$, $t_i \in \mathcal{A} \cup \{\mathfrak{P}\}$.*

A concrete trace is constructed recursively on a given path. If σ is a single state, $\mathsf{tr}(\sigma)$ is empty. Now, consider the path $\sigma = \sigma' \circ last(\sigma)\ l_k\ s_k$ where $\mathsf{tr}(\sigma')$ is known. The concrete trace of the path σ is constructed as following:

- $\mathsf{tr}(\sigma) = \mathsf{tr}(\sigma')\ \mathfrak{P}$ *if* $l_k \in]0,1]$
- $\mathsf{tr}(\sigma) = \mathsf{tr}(\sigma')\ l_k$ *if* $l_k \in \mathcal{A}$

Definition C.3 (Abstract trace)**.** *Given an IPC $P = \langle S, \mathcal{A}, \longrightarrow, \rightsquigarrow, s_0\rangle$, the abstract trace of a path σ is a sequence $\mathsf{tr}_\tau(\sigma) = t_1\ t_2\ \ldots\ t_m$ where for every $i \leq m$, $t_i \in \mathcal{A} \cup \{\mathfrak{P}\}$.*

An abstract trace is constructed recursively on a given path. If σ is a single state, $\mathsf{tr}(\sigma)$ is empty. Now, consider the path $\sigma = \sigma' \circ last(\sigma)\ l_k\ s_k$ where $\mathsf{tr}(\sigma')$ is known. The abstract trace of the path σ is constructed as its concrete trace, but silent τ-transitions are not represented:

- $\mathsf{tr}_\tau(\sigma) = \mathsf{tr}_\tau(\sigma')$ *if* $l_k = \tau$ *and* $last(\sigma') \approx s_k$
- $\mathsf{tr}_\tau(\sigma) = \mathsf{tr}_\tau(\sigma')\ \mathfrak{P}$ *if* $l_k \in]0,1]$
- $\mathsf{tr}_\tau(\sigma) = \mathsf{tr}(\sigma')\ l_k$ *otherwise*

In a given state, a scheduler deterministically chooses one of the interactive transitions that can be fired. The following definition is a generalization of definition 5.13.

Definition C.4 (Scheduler)**.** *Given an urgency-cut time deterministic IPC $P = \langle S = S_I \cup S_P, \mathcal{A}, \longrightarrow, \rightsquigarrow, s_0\rangle$, a scheduler $\mathfrak{s}$ on P is the function:*

$$\begin{array}{lcl} Paths(s_0) & \mapsto & \longrightarrow \cup \{\Uparrow\} \\ \sigma & \to & \begin{cases} last(\sigma) \xrightarrow{a} s_a & \text{if } last(\sigma) \in S_I, \text{ for some } a \in \mathcal{A} \text{ and } s_a \in S \\ \Uparrow & \text{if } last(\sigma) \in S_P \end{cases} \end{array}$$

When the last state of the considered path is interactive, the scheduler chooses the next interactive transition among those interactive transitions that can be fired. When the last state of the considered path is probabilistic, the scheduler lets the next state be determined by the probabilistic distribution given by probabilistic transitions that can be taken (which is denoted by $\Uparrow$).

Given an IPC $P = \langle S, \mathcal{A}, \longrightarrow, \rightsquigarrow, s_0\rangle$, and a scheduler $\mathfrak{s}$ defined over $Paths(s_0)$, we define the IPC $\mathfrak{s}(P)$ as the IPC obtained by scheduling of P by $\mathfrak{s}$.

In proof of lemma 5.8, we need to consider extensions of paths by schedulers. Therefore, we define path expanders and silent path expanders.

Definition C.5 (Path expander)**.** *Given an IPC $P = \langle S, \mathcal{A}, \longrightarrow, \rightsquigarrow, s_0\rangle$ and a scheduler $\mathfrak{s}$ on P, the $\mathfrak{s}$-path expander $\xi_{\mathfrak{s}}$ is the function :*

$$\begin{array}{rcccl} \xi_{\mathfrak{s}}: & Paths(s_0) & \mapsto & Paths(s_0) \\ & \sigma & \to & \sigma \circ \mathfrak{s}(\sigma) \end{array}$$

For every integer value k and every path $\sigma \in Paths(s_0)$, we define $\xi_{\mathfrak{s}}^k(\sigma)$ the function $\overbrace{\xi_{\mathfrak{s}}(\xi_{\mathfrak{s}}(...}^{\text{k times}}(\sigma)))$. By convention, $\xi_{\mathfrak{s}}^0(\sigma) = \sigma$.

Definition C.6 (Silent path expander)**.** *Given an IPC $P = \langle S, \mathcal{A}, \longrightarrow, \rightsquigarrow, s_0\rangle$, a scheduler $\mathfrak{s}$ on P, and the $\mathfrak{s}$-path expander $\xi_{\mathfrak{s}}$, we define the silent $\mathfrak{s}$-path expander $\Xi_{\mathfrak{s}}$ as the function:*

$$
\begin{array}{rcl}
\Xi_\sigma : \; Paths(s_0) & \mapsto & Paths(s_0) \\
\sigma & \rightarrow & \xi_{\mathfrak{s}}^n(\sigma) \text{ for some } n \text{ such that:} \\
& & \left(\forall k < n\right) \quad \mathfrak{s}(\xi_{\mathfrak{s}}^k(\sigma)) \;=\; last(\xi_{\mathfrak{s}}^k(\sigma)) \xrightarrow{\tau} s_k \text{ with } s_k \in [last(c)]_{/_\approx} \\
& & \mathfrak{s}(\xi_{\mathfrak{s}}^n(\sigma)) \;=\; \begin{cases} last(\xi_{\mathfrak{s}}^n(\sigma)) \xrightarrow{a} s_a & \text{for some } a \in \mathcal{A} \text{ and } s_a \in S \\ & \text{with } s_a \notin [last(c)]_{/_\approx} \text{ if } a = \tau \\ \text{or} & \\ last(\xi_{\mathfrak{s}}^n(\sigma)) \overset{p_a}{\rightsquigarrow} s_a & \text{for some } p_a \in \,]0,1] \text{ and } s_a \in S \end{cases}
\end{array}
$$

For a given path σ and a given scheduler $\mathfrak{s}$, the silent $\mathfrak{s}$-path expander $\Xi_{\mathfrak{s}}$ gives the extension σ' of σ such that $last(\sigma) \approx last(\sigma')$ and $last(\sigma') \not\approx last(\xi_{\mathfrak{s}}(\sigma'))$. The silent-path expansion of a path always exists in non-Zeno time deterministic IPCs because τ-loops are prohibited by the non-Zeno property.

Using these definitions, we can link paths in branching bisimilar tdIPCs. More precisely, given a path in a tdIPC P, we can find a path in a branching equivalent IPC P' such that the first states and last states of those two paths are branching equivalent and the two paths are similar (in terms of trace).

Lemma C.1. *For every urgency-cut time deterministic IPCs $P = \langle S, \mathcal{A}, \longrightarrow, \rightsquigarrow, s_0\rangle$ and $P' = \langle S', \mathcal{A}', \longrightarrow, \rightsquigarrow, s_0'\rangle$, $P \approx P'$ implies that for every path σ in $Paths(s_0)$, there is a path σ' in $Paths(s_0')$ such that $last(\sigma) \approx last(\sigma')$ and $\mathrm{tr}_\tau(\sigma) = \mathrm{tr}_\tau(\sigma')$.*

Proof. Let n be the length of the path σ. The proof is done by induction on n.

- if $n = 0$, $\sigma = s_0$. Choosing $\sigma' = s_0'$, we have $s_0 \approx s_0'$ and $\mathrm{tr}_\tau(\sigma) = \mathrm{tr}_\tau(\sigma') = \emptyset$. Consequently, lemma C.1 is verified.
- Suppose that we have $\sigma = \tilde{\sigma} \circ last(\tilde{\sigma})\; l\; s_{n+1}$ with $\tilde{\sigma}$ a path of length n satisfying lemma C.1. We have to show that lemma C.1 is verified for σ.
  The induction assumption gives a path $\tilde{\sigma}' \in Paths(s_0')$ having the same trace as $\tilde{\sigma}$ and such that $last(\tilde{\sigma}')$ and $last(\tilde{\sigma})$ are branching equivalent, i.e., $last(\tilde{\sigma}') \approx last(\tilde{\sigma})$. Call $s_n = last(\tilde{\sigma})$ and $s_n' = last(\tilde{\sigma}')$.
  (i) either $l \in \mathcal{A}$. Thus we have an interactive transition $s_n \xrightarrow{l} s_{n+1}$ from s_n to s_{n+1}. From $s_n \approx s_n'$, we deduce:
  - ◇ either $l = \tau$ and $s_n' \approx s_{n+1}$. We can take $\sigma' = \tilde{\sigma}'$ and lemma C.1 is satisfied.
  - ◇ or $s_n' \xrightarrow{\tau^*} \tilde{s}_n'$ for some $\tilde{s}_n'$ such that $\tilde{s}_n' \approx s_n'$ and $\gamma_0(\tilde{s}_n', l, [s_{n+1}]_{/_\approx}) = \gamma_0(s_n, l, [s_{n+1}]_{/_\approx})$. Thus, there is an interactive transition labeled l from $\tilde{s}_n'$ to a state, say s_{n+1}', in the class of s_{n+1}. Consequently, we found a state s_{n+1}' that is equivalent to s_{n+1} and reachable from s_n' with $s_n' \xrightarrow{\tau^*} \tilde{s}_n' \xrightarrow{l} s_{n+1}'$. Considering the path $\sigma = \tilde{\sigma}' \circ s_n' \xrightarrow{\tau^*} \tilde{s}_n' \xrightarrow{l} s_{n+1}'$, lemma C.1 is satisfied.

(ii) or $l \in]0,1]$. Thus we have a probabilistic transition $s_n \overset{l}{\rightsquigarrow} s_{n+1}$. Consider the equivalence class $\mathcal{C} = [s_{n+1}]_{/\approx}$. This class satisfies $\gamma_P(s_n,\mathcal{C}) > 0$. Because P is urgency-cut, we have that $s_n \overset{\tau}{\not\longrightarrow}$. $s_n \approx s'_n$ implies that $s'_n \overset{\tau*}{\longrightarrow} \tilde{s}'_n$ for some $\tilde{s}'_n$ such that $\tilde{s}'_n \approx s'_n$ and for every equivalence class $\mathcal{C}$, $\gamma_P(\tilde{s}'_n,\mathcal{C}) = \gamma_P(s_n,\mathcal{C}) > 0$. Thus, there is at least one probabilistic transition from $\tilde{s}'_n$ to a state of $\mathcal{C}$. Call s'_{n+1} this state and l' the probability from $\tilde{s}'_n$ to s'_{n+1}. We found a state s'_{n+1} that is equivalent to s_{n+1} and reachable from s'_n, i.e., $s'_n \overset{\tau*}{\longrightarrow} \tilde{s}'_n \overset{l}{\rightsquigarrow} s'_{n+1}$. Considering the path $\sigma = \tilde{\sigma}' \circ s'_n \overset{\tau*}{\longrightarrow} \tilde{s}'_n \overset{l}{\rightsquigarrow} s'_{n+1}$, lemma C.1 is verified.

□

Finally, we prove that two branching bisimilar urgency-cut tdIPCs can be scheduled to obtain two branching bisimilar dIPCs.

Lemma C.2. *Given two urgency-cut tdIPCs $P = \langle S, \mathcal{A}, \longrightarrow, \rightsquigarrow, s_0\rangle$ and $P' = \langle S', \mathcal{A}', \longrightarrow, \rightsquigarrow, s'_0\rangle$ such that P and P' are branching equivalent, for every scheduler $\mathfrak{s}$ over $Paths(s_0)$ in P, there exists a scheduler $\mathfrak{s}'$ over $Paths(s'_0)$ in P' such that $\mathfrak{s}(P)$ and $\mathfrak{s}'(P')$ are branching equivalent, $\mathfrak{s}(P) \approx \mathfrak{s}'(P')$.*

Proof. Given P, consider a scheduler $\mathfrak{s}$ over $Paths(s_0)$. We are going to construct a scheduler $\mathfrak{s}'$ over $Paths(s'_0)$ in P' such that the branching equivalence between $\mathfrak{s}(P)$ and $\mathfrak{s}'(P')$ is preserved.

We construct $\mathfrak{s}'$ considering every path $\sigma' \in Paths(s'_0)$ as follows :

For every path $\sigma' \in Paths(s'_0)$, lemma C.1 gives us a path $\sigma \in Paths(s_0)$ such that $last(\sigma') \approx last(\sigma)$. Consider the silent $\mathfrak{s}$-path expansion $\Xi_{\mathfrak{s}}(\sigma)$ of σ.

- either $\mathfrak{s}(\Xi_{\mathfrak{s}}(\sigma)) = last(\Xi_{\mathfrak{s}}(\sigma)) \overset{a}{\longrightarrow} s_a$ for some $a \in \mathcal{A}$ and $s_a \in S$. We have by definition C.6 of $\Xi_{\mathfrak{s}}$ that $last(\Xi_{\mathfrak{s}}(\sigma)) \approx last(\sigma)$. Call $\tilde{s}$ the state $last(\Xi_{\mathfrak{s}}(\sigma))$.
  To preserve the branching equivalence between $\mathfrak{s}(P)$ and $\mathfrak{s}'(P')$, we have to find a state $\tilde{s}'$ such that $last(\sigma') \overset{\tau*}{\longrightarrow} \tilde{s}'$, $\tilde{s}' \approx \tilde{s}$ and $\gamma_0(\tilde{s}', a, [s_a]_{/\approx})$ is true. Because $\gamma_0(\tilde{s}, a, [s_a]_{/\approx})$ is true, the branching equivalence between $last(\sigma')$ and $\tilde{s}$ in P' and P implies that $\gamma_0(\tilde{s}', a, [s_a]_{/\approx})$ is true for some $\tilde{s}' \in S'$ such that $last(\sigma') \overset{\tau*}{\longrightarrow} \tilde{s}'$ and $\tilde{s}' \approx \tilde{s}$.
  $\gamma_0(\tilde{s}', a, [s_a]_{/\approx})$ ensures the existence of a state s'_a branching equivalent to s_a and such that $\tilde{s}' \overset{a}{\longrightarrow} s'_a$. In this case, the branching equivalence between $\mathfrak{s}(P)$ and $\mathfrak{s}(P')$ is preserved taking $\mathfrak{s}'(\sigma') = \tilde{s}' \overset{a}{\longrightarrow} s'_a$.
- or $\mathfrak{s}(\Xi_{\mathfrak{s}}(\sigma)) = last(\Xi_{\mathfrak{s}}(\sigma)) \overset{p}{\rightsquigarrow} s_p$ for some $p \in]0,1]$ and $s_p \in S$. We have by definition C.6 of $\Xi_{\mathfrak{s}}$ that $last(\Xi_{\mathfrak{s}}(\sigma)) \approx last(\sigma)$. Call $\tilde{s}$ the state $last(\Xi_{\mathfrak{s}}(\sigma))$.
  To preserve the branching equivalence between $\mathfrak{s}(P)$ and $\mathfrak{s}'(P')$, and because P is urgency cut (we have $last(\Xi_{\mathfrak{s}}(\sigma)) \overset{\tau}{\not\longrightarrow}$), we have to find state $\tilde{s}'$ such that $last(\sigma') \overset{\tau*}{\longrightarrow} \tilde{s}'$, $\tilde{s}' \approx \tilde{s}$ and $\gamma_P(s', \left[s_p\right]_{/\approx}) = \gamma_P(\tilde{s}, \left[s_p\right]_{/\approx})$. The branching equivalence between $last(\sigma')$ and $\tilde{s}$ in P' and P implies that $\gamma_P(\tilde{s}', \left[s_p\right]_{/\approx}) = \gamma_P(\tilde{s}, \left[s_p\right]_{/\approx})$ for some $\tilde{s}' \in S'$ such that $last(\sigma') \overset{\tau*}{\longrightarrow} \tilde{s}'$ and $\tilde{s}' \approx \tilde{s}$.
  Thus, the branching equivalence between $\mathfrak{s}(P)$ and $\mathfrak{s}'(P')$ is preserved taking $\mathfrak{s}'(\sigma') = \Uparrow$.

□

C.2 DTAMCs Associated to Branching Bisimilar Deterministic Interactive Probabilistic Chains

In this section, we prove that the DTAMCs associated to branching bisimilar dIPCs are strongly bisimilar. Let $\mathscr{D}$ be the set of all DTAMCs. We first introduce several lemmas.

Lemma C.3. *Let $P = \langle S = S_I \cup S_P, \mathcal{A}, \longrightarrow, \rightsquigarrow, s_0\rangle$ be an urgency cut dIPC over $\mathcal{A}$, and k be the length of the longest sequence of interactive transitions in P. For all states s_1', s_1, s_2 in S and every $e_1 \in \mathfrak{E}$ such that $s_1 \approx s_2$ and $s_1' \xrightarrow{e_1} s_1$, we have:*

$$\left(\forall s_2' \in S\right)\left(\forall e_2 \in \mathfrak{E}\right) \quad s_2' \xrightarrow{e_2} s_2 \wedge \mathfrak{S}(e_2) = \mathfrak{S}(e_1) \Longrightarrow s_2' \approx s_1'$$

Proof. By induction on the length n of the word defined by the regular expression e_1.

If $n = 0$, i.e., $s_1' \xrightarrow{\mathfrak{e}} s_1$ (i.e., $s_1' = s_1$), the lemma is clearly verified.

Suppose that $s_1' \xrightarrow{e_{n+1}} s_1$ with e_{n+1} the regular expression defining a word $a_{n+1}\ a_n\ \ldots\ a_1$ (of length $n+1$). In other words, there is a state $s_1'' \in S$ such that $s_1' \xrightarrow{a_{n+1}} s_1'' \xrightarrow{e_n} s_1$ with e_n the regular expression defining the word $a_n\ a_{n-1}\ \ldots\ a_1$ (of length n).

Suppose also that $s_2' \xrightarrow{e'_{n+1}} s_2$, with $\mathfrak{S}(e'_{n+1}) = \mathfrak{S}(e_{n+1})$, e'_{n+1} being a regular expression. The word defined by e'_{n+1} has the form $\tau^*\ a_{n+1}\ a_m\ a_{m-1}\ \ldots\ a_1$ with $a_m\ a_{m-1}\ \ldots\ a_1$ a word defined by a regular expression e'_n that satisfies $\mathfrak{S}(e'_n) = \mathfrak{S}(e_n)$. There is consequently a state s_2'' such that $s_2' \xrightarrow{\tau^*} \xrightarrow{a_{n+1}} s_2'' \xrightarrow{e'_n} s_2$.

The induction assumption gives us

$$s_2'' \xrightarrow{e'_n} s_2 \wedge \mathfrak{S}(e'_n) = \mathfrak{S}(e_n) \Longrightarrow s_2'' \approx s_1''$$

Because the IPC P is deterministic, we can finally conclude that $s_2' \approx s_1'$. Indeed, s_1' and s_2' can only fire the same action a_{n+1} after a potentially empty sequence of τ transitions, to reach bisimilar states s_1'' and s_2''. □

Lemma C.4. *Let $P = \langle S = S_I \cup S_P, \mathcal{A}, \longrightarrow, \rightsquigarrow, s_0\rangle$ be an urgency cut dIPC over $\mathcal{A}$, and k be the length of the longest sequence of interactive transitions in P. For every pair of states s_1 and s_2 in S satisfying $s_1 \approx s_2$, we have:*

$$\left(\forall s_1' \in S_P\right)\left(\forall e_1 \in \mathfrak{E}\right) \quad s_1 \xrightarrow{e_1} s_1' \Longrightarrow \left(\left(\exists s_2' \in S_P\right)\left(\exists e_2 \in \mathfrak{E}\right) \quad s_2 \xrightarrow{e_2} s_2' \wedge \mathfrak{S}(e_1) = \mathfrak{S}(e_2) \wedge s_1' \approx s_2'\right)$$

Proof. By induction on the length of the word defined by the regular expression e_1, it is a direct application of the IPC branching bisimulation (definition 5.8). □

Finally, we can prove that the DTAMCs associated to branching bisimilar urgency cut dIPCs are strongly bisimilar.

Lemma C.5. *Let $P = \langle S = S_I \cup S_P, \mathcal{A}, \longrightarrow, \rightsquigarrow, s_0\rangle$ be an urgency cut dIPC over $\mathcal{A}$, k be the length of the longest sequence of interactive transitions in P, and $\mathcal{M}(P) = \langle C, \rightsquigarrow, c_0, \mathscr{A}\rangle$ be the DTAMC associated to P. Consider two states $s_1, s_2 \in S_P$. For every multiset $\mathscr{S} \in \mathcal{P}^{\times}_{\leq k}(\mathcal{A})$, we have:*

$$s_1 \approx s_2 \Longrightarrow \langle s_1, \mathscr{S}\rangle \sim_p \langle s_2, \mathscr{S}\rangle$$

Proof. Consider two states s_1 and s_2 in S_P such that $s_1 \approx s_2$ and a multiset $\mathcal{S} \in \mathcal{P}^{\times}_{\leq k}(\mathcal{A})$. The states $\langle s_1, \mathcal{S}\rangle$ and $\langle s_2, \mathcal{S}\rangle$ in C are strongly bisimilar, $\langle s_1, \mathcal{S}\rangle \sim_p \langle s_2, \mathcal{S}\rangle$, if there exists a relation $\mathcal{R}$ that is a strong bisimulation and such that $\langle s_1, \mathcal{S}\rangle \ \mathcal{R} \ \langle s_2, \mathcal{S}\rangle$. We are going to construct such a relation $\mathcal{R}$.

Let $\mathcal{R} = \left\{(\langle s_1, \mathcal{S}\rangle, \langle s_2, \mathcal{S}\rangle) \mid \mathcal{S} \in \mathcal{P}^{\times}_{\leq k}(\mathcal{A}),\ s_1, s_2, \in S_P,\ s_1 \approx s_2\right\}$. $\mathcal{R}$ is clearly an equivalence relation, because $\approx$ is an equivalence relation.

Consider an equivalence class $\mathcal{C}_{\mathcal{M}} \in \mathscr{D}/_{\mathcal{R}}$. We can distinguish two cases with respect to the multiset inducing the class $\mathcal{C}_{\mathcal{M}}$:

- Either there is no state $\langle s, \mathcal{S}\rangle \in \mathcal{C}_{\mathcal{M}}$, such that a path $s_1 \overset{p_1}{\rightsquigarrow} s_1' \xrightarrow{e_1} s$ exists in P, for some $p_1 > 0$, $s_1' \in S$, and e_1 satisfying $\mathfrak{S}(e_1) = \mathcal{S}$. According to lemma C.4, we can prove by contraposition that there is thus no state $\langle s', \mathcal{S}\rangle \in \mathcal{C}_{\mathcal{M}}$, such that a path $s_2 \overset{p_2}{\rightsquigarrow} s_2' \xrightarrow{e_2} s'$ exists in P, for som $p_2 > 0$, $s_2' \in S$, and e_2 satisfying $\mathfrak{S}(e_1) = \mathcal{S}$. Consequently, we have:

$$\gamma_P\left(\langle s_1, \mathcal{S}\rangle, \mathcal{C}_{\mathcal{M}}\right) = \gamma_P\left(\langle s_2, \mathcal{S}\rangle, \mathcal{C}_{\mathcal{M}}\right) = 0$$

- Or there is at least one state $\langle s, \mathcal{S}\rangle \in \mathcal{C}_{\mathcal{M}}$, such that a path $s_1 \overset{p_1}{\rightsquigarrow} s_1' \xrightarrow{e_1} s$ exists in P, for some $p_1 > 0$, $s_1' \in S$ and e_1 satisfying $\mathfrak{S}(e_1) = \mathcal{S}$.
  Consider the set of states

$$\overline{\mathcal{S}_1} = \left\{s_i' \in S \mid \left(\exists s_i \in S_P\right)\left(\exists e_i \in \mathfrak{E}\right)\left(\exists p_i > 0\right) \quad s_1 \overset{p_i}{\rightsquigarrow} s_i' \xrightarrow{e_i} s_i \wedge \langle s_i, \mathfrak{S}(e_i)\rangle \in \mathcal{C}_{\mathcal{M}}\right\}$$

  By definition of $\mathcal{R}$, Forall all $s_i' \in \overline{\mathcal{S}_1}$ such that $s_1 \overset{p_i}{\rightsquigarrow} s_i' \xrightarrow{e_i} s_i$ (for some p_i, e_i and s_i), we have that $s_i \approx s \wedge \mathfrak{S}(e_i) = \mathcal{S}$. According to lemma C.3, we have that, for all states $s_i, s_j \in \overline{\mathcal{S}_1}$, $s_i \approx s_j$, that implies that $\overline{\mathcal{S}_1}$ is a subset of an equivalence class $\mathcal{C}_p$ of $\mathscr{P}/_{\approx}$.
  In addition, $\overline{\mathcal{S}_1}$ is the smallest subset of $\mathcal{C}_p$ such that $\gamma_P\left(s_1, \overline{\mathcal{S}_1}\right) = \gamma_P\left(s_1, \mathcal{C}_p\right)$. Indeed, suppose that $s_1 \overset{p'}{\rightsquigarrow} s_1''$ for some $s_1'' \in \mathcal{C}_p \backslash \overline{\mathcal{S}_1}$. We have $s_1'' \approx s_1'$ (because $s_1'' \in \mathcal{C}_p \backslash \overline{\mathcal{S}_1}$) and $s_1' \xrightarrow{e_1} s$. According to lemma C.4, $\left(\exists s' \in S_P\right)\left(\exists e_1' \in \mathfrak{E}\right)$ such that $s_1'' \xrightarrow{e_1'} s'$ with $s' \approx s$ and $\mathfrak{S}(e_1') = \mathfrak{S}(e_1)$. Thus, $s_1'' \in \overline{\mathcal{S}_1}$, which is in contradiction with $s_1'' \in \mathcal{C}_p \backslash \overline{\mathcal{S}_1}$.
  Now, Consider the set of states

$$\overline{\mathcal{S}_2} = \left\{s_j' \in S \mid \left(\exists s_j \in S_P\right)\left(\exists e_j \in \mathfrak{E}\right)\left(\exists p_j > 0\right) \quad s_2 \overset{p_j}{\rightsquigarrow} s_j' \xrightarrow{e_j} s_j \wedge \langle s_j, \mathfrak{S}(e_j)\rangle \in \mathcal{C}_{\mathcal{M}}\right\}$$

  By definition of $\mathcal{R}$, we have that $\left(\forall j\right)\ s_j \approx s \wedge \mathfrak{S}(e_j) = \mathcal{S}$. According to lemma C.3, one can say that $\overline{\mathcal{S}_2}$ is also a subset of $\mathcal{C}_p$, and following the same reasonning than for $\overline{\mathcal{S}_1}$, $\overline{\mathcal{S}_2}$ is the smallest subset of $\mathcal{C}_p$ such that $\gamma_P\left(s_2, \overline{\mathcal{S}_2}\right) = \gamma_P\left(s_2, \mathcal{C}_p\right)$.
  Because s_1 and s_2 are branching bisimilar ($s_1 \approx s_2$), we have $\gamma_P\left(s_1 \mathcal{C}_p\right) = \gamma_P\left(s_2, \mathcal{C}_p\right)$, which induces $\gamma_P\left(s_1, \overline{\mathcal{S}_1}\right) = \gamma_P\left(s_2, \overline{\mathcal{S}_2}\right)$, and finally

$$\gamma_p\left(\langle s_1, \mathcal{S}\rangle, \mathcal{C}_{\mathcal{M}}\right) = \gamma_P\left(\langle s_2, \mathcal{S}\rangle, \mathcal{C}_{\mathcal{M}}\right)$$

□

C.3 Associated DTAMCs of Probabilistic Branching Bisimilar Time Deterministic Interactive Probabilistic Chains

Theorem C.1. *Associated DTAMCs of probabilistic branching bisimilar tdIPCs are strongly bisimilar, i.e.,*

$$\left(\forall\,(P_1,P_2)\in\mathscr{P}^2\right)\quad\left(P_1\approx P_2\right)\Longrightarrow\left(\mathcal{M}(P_1)\sim_p\mathcal{M}(P_2)\right)$$

Proof. Consider two time deterministic IPCs P_1 and P_2, such that $P_1\approx P_2$, and a scheduler $\mathscr{s}_1$ such that $\mathscr{s}_1(P_1)$ is the deterministic IPC obtained by scheduling of P_1.

According to lemma C.2, we can find a scheduler $\mathscr{s}_2$ such that $\mathscr{s}_1(P_1)\approx\mathscr{s}_2(P_2)$.

According to lemma 5.7, the DTAMC associated to P_1 is strongly bisimilar to the DTAMC associated to $\mathscr{s}(P_1)$, i.e., $\mathcal{M}(P_1)\sim_p\mathcal{M}(\mathscr{s}(P_1))$. Similarly, $\mathcal{M}(P_2)\sim_p\mathcal{M}(\mathscr{s}(P_2))$.

Finally, according to lemma C.5, the DTAMC associated to two bisimilar urgency-cut dIPCs are strongly bisimilar. Thus, we have $\mathcal{M}(\mathscr{s}(P_1))\sim_p\mathcal{M}(\mathscr{s}(P_2))$.

By transitivity of the branching bisimulation equivalence, we can conclude that $\mathcal{M}(P_1)\sim_p\mathcal{M}(P_2)$. □

Appendix D

Relationship between SMCs and DTAMCs

In this appendix, we prove lemma 5.9, i.e., we prove that the steady state probabilities of a DTAMC associated to an IPC are related to the steady state probabilities of the SMC associated to the DTAMC.

Definition D.1 (k-scaled DTMC)**.** *Consider an ergodic DTMC $\mathcal{M}_P = \langle S, \rightsquigarrow, \overline{s}\rangle$. The k-scaled DTMC $\mathcal{M}_P^k$ of $\mathcal{M}_P$ is the DTMC $\mathcal{M}_P^k = \langle S', \rightsquigarrow, \overline{s}\rangle$ satisfying:*

- *the state space S is a subset of the state space S', i.e., $S \subseteq S'$*
- *For every transition $s_1 \overset{p}{\rightsquigarrow} s_2$ (for some $s_1, s_2 \in S$ and $p \in]0,1]$) in $\mathcal{M}_P$, there is a sequence of k probabilistic transitions in $\mathcal{M}_P^k$: $s_1 \overset{p}{\rightsquigarrow} s_{1,1,2} \overset{1}{\rightsquigarrow} s_{1,2,2} \overset{1}{\rightsquigarrow} \cdots \overset{1}{\rightsquigarrow} s_{1,k-1,2} \overset{1}{\rightsquigarrow} s_2$, with $\{s_{1,i,2}\}_{i<k} \subset S^k \backslash S$.*

Informally, the k-scaled DTMC of $\mathcal{M}_P$ is obtained by splitting each transition of $\mathcal{M}_P$ in k successive transitions. We introduce some notations concerning the state space S' of a k-scaled DTMC. Consider a DTMC $\mathcal{M}_P = \langle S, \rightsquigarrow, \overline{s}\rangle$ and the k-scaled DTMC $\mathcal{M}_P^k = \langle S', \rightsquigarrow, \overline{s}\rangle$ of $\mathcal{M}_P$. For every state $s_i \in S$, $S_{\rightsquigarrow s_i}$ is the set of predecessors of s_i in $\mathcal{M}_P$. For every state $s_i \in S \subset S'$, and every $k' < k$ we note $S_{s_i,k-k'} \subset S'$ the set of states:

$$S_{s_i,k-k'} = \left\{ s' \in S' \backslash S \;\middle|\; s' \underbrace{\overset{1}{\rightsquigarrow} \cdots \overset{1}{\rightsquigarrow}}_{k-k' \text{ transitions}} s_i \right\}$$

Consequently, the state space S' satisfies:

$$S' = S \cup \left(\bigcup_{s_i \in S} \left(\bigcup_{k'<k} S_{s_i,k-k'} \right) \right)$$

We illustrate the subdivision of the state space S' of a k-scaled DTMC in figure D.1. We introduce the notation $Paths_{\mathcal{M}_P}(s)$ to denote the set of paths starting in state s in the DTMC $\mathcal{M}_P$.

We first consider a lemma introducing a link between the steady state probability of a state s_i and the steady state probabilities of states in a set $S_{s_i,l}$ of $\mathcal{M}_P^k$.

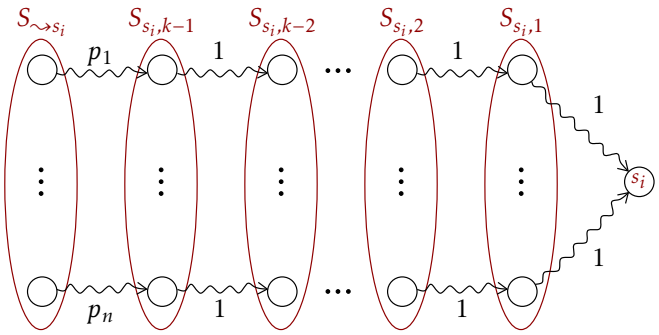


Figure D.1: Particular sets of states of a k-scaled DTMC

Lemma D.1. *Consider an ergodic DTMC $\mathcal{M}_P = \langle S, \rightsquigarrow, \overline{s}\rangle$ and the k-scaled DTMC $\mathcal{M}_P^k$ of $\mathcal{M}_P$, $\mathcal{M}_P^k = \langle S', \rightsquigarrow, \overline{s}\rangle$. Let π' be the steady-state probability vector of $\mathcal{M}_P^k$. For every state $s_i \in S \subseteq S'$ and every $k' < k$, we have:*

$$\pi'_{s_i} = \sum_{s \in S_{s_i,k-k'}} \pi'_s$$

Proof. The lemma is clearly satisfied for the the set of state $S_{s_i,1}$ (i.e., for $k' = k-1$). By induction, it is also satisfied for other values $k' < k$, because $\sum_{s\in S_{s_i,k-k'}} \pi'_s = \sum_{s\in S_{s_i,k-k'+1}} \pi'_s$ □

For every value k, the steady state probabilities of a DTMC $\mathcal{M}_P$ and of the k-scaled DTMC $\mathcal{M}_P^k$ of $\mathcal{M}_P$ are related.

Lemma D.2 (steady state probabilities of a k-scaled DTMC)**.** *Consider a DTMC $\mathcal{M}_P = \langle S, \rightsquigarrow, \overline{s}\rangle$ and the k-scaled DTMC $\mathcal{M}_P^k = \langle S', \rightsquigarrow, \overline{s}\rangle$ of $\mathcal{M}_P$. Let π be the steady-state probability vector of $\mathcal{M}_P$, and π' be the steady-state probability vector of $\mathcal{M}_P^k$. For all state $s_i \in S$, and all $k' < k$, we have:*

$$\pi_{s_i} = k \times \sum_{s_j \in S_{s_i,k-k'}} \pi'_{s_j}$$

Proof. According to lemma D.1, we can write that:

$$\left(\forall s_i \in S\right), \quad \sum_{s \in \{s_i\} \cup \bigcup_{k'<k} S_{s_i,k-k'}} \pi'_s = \pi'_{s_i} + \sum_{k'<k} \sum_{s\in S_{s_i,k-k'}} \pi'_s = k\, \pi'_{s_i} \tag{1}$$

Because the steady state probability vector is a probabilistic distribution, we have that

$$\sum_{s\in S} \pi_s = 1 \tag{2}$$

and

$$\sum_{s\in S'} \pi'_s = 1 \tag{3}$$

From equation 1 and 3, we can deduce that

$$k \sum_{s\in S} \pi'_s = 1 \tag{4}$$

For every pair of state $(s_1, s_2) \in S^2$, let $p_{s_1 s_2}$ be the transition probability from s_1 to s_2 in $\mathcal{M}_P$ ($p_{s_1 s_2}$ is equal to zero if there is no probabilistic transition from s_1 to s_2). In addition to equation 2, the steady state probabilities in $\mathcal{M}_P$ satisfy:

$$\Big(\forall s \in S\Big) \quad \pi_s = \sum_{s' \in S} p_{s's}\, \pi'_s \tag{5}$$

By construction, and in addition to equation 3, the steady state probabilities of states in S in $\mathcal{M}_P^k$ satisfy:

$$\Big(\forall s \in S\Big) \quad \pi'_s = \sum_{s' \in S} p_{s's}\, \pi'_s \tag{6}$$

In an ergodic DTMC, the system of equations given by the set of equations 5 (resp. equations 6) is linearly dependent and has an infinity of linearly dependent solutions; a unique solution is ensured by additionally considering equation 2 (resp. equation 3), because we are interested in the single probabilistic distribution solution.

However, because systems of equations given by the set of equations 5 and 6 are identical (regardless of the variable names),we can write that for each state $s_i \in S$ and for every states $s_j \in S$, there exists $\psi_{s_i \rightsquigarrow s_j}$ such that:

$$\pi_{s_j} = \psi_{s_i \rightsquigarrow s_j}\, \pi_{s_i} \quad \text{and} \quad \pi'_{s_j} = \psi_{s_i \rightsquigarrow s_j}\, \pi'_{s_i}$$

From equations 2 and 3, we can thus write:

$$\begin{aligned}
& \sum_{s \in S} \pi_s = \sum_{s \in S'} \pi'_s \\
\iff & \sum_{s \in S} \pi_s = \sum_{s \in S} \pi'_s + \sum_{s \in S} \sum_{k'=1}^{k-1} \underbrace{\sum_{s_j \in S_{s,k-k'}} \pi'_{s_j}}_{\pi'_s \text{ according to lemma D.1}} \\
\iff & \sum_{s \in S} \pi_s = \sum_{s \in S} \pi'_s + \sum_{s \in S} \Big((k-1)\, \pi'_s\Big) \\
\iff & \sum_{s \in S} \pi_s = \sum_{s \in S} \pi'_s + (k-1) \sum_{s \in S} \pi'_s \\
\iff & \sum_{s \in S} \pi_s = k \sum_{s \in S} \pi'_s
\end{aligned}$$

Now consider a state $s_i \in S$. We can write:

$$\begin{aligned}
& \sum_{s \in S} \pi_s = k \sum_{s \in S} \pi'_s \\
\iff & \sum_{s \in S} \psi_{s_i \rightsquigarrow s}\, \pi_{s_i} = k \sum_{s \in S} \psi_{s_i \rightsquigarrow s}\, \pi'_{s_i} \\
\iff & \pi_{s_i} \left(\sum_{s \in S} \psi_{s_i \rightsquigarrow s} \right) = k\, \pi'_{s_i} \left(\sum_{s \in S} \psi_{s_i \rightsquigarrow s} \right) \\
\iff & \pi_{s_i} = k\, \pi'_{s_i}
\end{aligned}$$

Finally the lemma is satisfied because of lemma D.1.

□

Lemma D.3 (Steady-state probabilities of an SMC and its embedded DTMC)**.** *Consider an SMC* $\mathcal{M}_S = \langle S, \mathcal{A}, \boxdot\!\!\rightsquigarrow, \mathcal{D}, \overline{s}\rangle$ *such that there exists* $k \in \mathbb{N}$ *such that:*

$$\left(\forall s_1, s_1' \in S\right)\left(\forall a \in \mathcal{A}\right)\left(\forall p \in]0,1]\right) \quad \left(s_1 \overset{a}{\boxdot\!\!\rightsquigarrow}{}^{p}\ s_1'\right) \Longrightarrow \left(\mathcal{D}\left(s_1 \overset{a}{\boxdot\!\!\rightsquigarrow}{}^{p}\ s_1'\right) = \mathbf{C}_k\right)$$

Let $\mathcal{M}_P = \langle S, \rightsquigarrow, \overline{s}\rangle$ *be the embedded DTMC of* $\mathcal{M}_S$*. Let* π *be the steady-state probability vector of* $\mathcal{M}_P$*, and* π' *the steady-state probability vector of* $\mathcal{M}_S$*. Then,* π *and* π' *satisfy* $\pi = \pi'$*.*

Proof. Consider a state $s_i \in S$. According to definition 2.39, the steady state probability π'_{s_i} of the state s_i in $\mathcal{M}_S$ is:

$$\pi'_{s_i} = \frac{\pi_{s_i} E\left[\mathcal{D}_{\mathsf{SJ}}(s_i,.)\right]}{\sum_{s_j \in S}\left(\pi_{s_j} E\left[\mathcal{D}_{\mathsf{SJ}}(s_j,.)\right]\right)}$$

Because the distribution associated to all transitions is $\mathbf{C}_k$, the distribution of the sojourn time in each state is also $\mathbf{C}_k$. Consequently, for every state $s \in S$, the expected value $E\left[\mathcal{D}_{\mathsf{SJ}}(s,.)\right]$ of the distribution of the sojourn time in s is equal to k. We can thus directly conclude that $\pi'_{s_i} = \pi_{s_i}$. □

Lemma D.4 (Aggregation of transitions in an SMC)**.** *Consider an SMC* $\mathcal{M}_S^{(1)} = \langle S \cup \{s\}, \mathcal{A}, \boxdot\!\!\rightsquigarrow, \mathcal{D}^{(1)}, \overline{s}\rangle$ *such that the state s satisfies:*

- *there is a single incoming transition in state s,* $s_0 \overset{\tau}{\boxdot\!\!\rightsquigarrow}{}^{p}\ s$ *(*$s_0 \in S$*, and* $p \in]0,1]$*)*
- *there is a single outgoing transition from state s,* $s \overset{\tau}{\boxdot\!\!\rightsquigarrow}{}^{1}\ s_1$ *(*$s_1 \in S$*)*
- $\left(\exists n \in \mathbb{N}_+\right) \quad \mathcal{D}^{(1)}\left(s_0 \overset{\tau}{\boxdot\!\!\rightsquigarrow}{}^{p}\ s\right) = \mathbf{C}_n$
- $\mathcal{D}^{(1)}\left(s \overset{\tau}{\boxdot\!\!\rightsquigarrow}{}^{1}\ s_1\right) = \mathbf{C}_1$

Consider also a second SMC $\mathcal{M}_S^{(2)} = \langle S, \mathcal{A}, \boxdot\!\!\rightsquigarrow, \mathcal{D}^{(2)}, \overline{s}\rangle$ *constructed from* $\mathcal{M}_S^{(1)}$*, and satisfying:*

- *for all transition* $s_i \overset{a}{\boxdot\!\!\rightsquigarrow}{}^{p'}\ s_j$ *(*$a \in \mathcal{A}$ *and* $p' \in]0,1]$*) in* $\mathcal{M}_S^{(1)}$ *such that* $s_i, s_j \in S$*,* $s_i \neq s$*, and* $s_j \neq s$*, there is the same transition* $s_i \overset{a}{\boxdot\!\!\rightsquigarrow}{}^{p'}\ s_j$ *in* $\mathcal{M}_S^{(2)}$*, and* $\mathcal{D}^{(1)}\left(s_i \overset{a}{\boxdot\!\!\rightsquigarrow}{}^{p'}\ s_j\right) = \mathcal{D}^{(2)}\left(s_i \overset{a}{\boxdot\!\!\rightsquigarrow}{}^{p'}\ s_j\right)$
- *there is a transition* $s_0 \overset{\tau}{\boxdot\!\!\rightsquigarrow}{}^{p}\ s_1$ *in* $\mathcal{M}_S^{(2)}$ *that satisfies* $\mathcal{D}^{(2)}\left(s_0 \overset{\tau}{\boxdot\!\!\rightsquigarrow}{}^{p}\ s_1\right) = \mathbf{C}_{n+1}$*.*

Let $\pi^{(1)'}$ *be the steady state probability vector of* $\mathcal{M}_S^{(1)}$ *and* $\pi^{(2)'}$ *be the steady state probability vector of* $\mathcal{M}_S^{(2)}$*. Then,* $\pi^{(1)'}$ *and* $\pi^{(2)'}$ *satisfy:*

$$\left(\forall s \in S \backslash \{s_0\}\right) \quad \pi_s^{(2)'} = \pi_s^{(1)'}$$

and

$$\pi_{s_0}^{(2)'} = \pi_{s_0}^{(1)'} + \pi_s^{(1)'}$$

Proof. Call $\mathcal{M}^{(1)}$ the embedded DTMC of $\mathcal{M}_S^{(1)}$ and $\mathcal{M}^{(2)}$ the embedded DTMC of $\mathcal{M}_S^{(2)}$. $\mathcal{M}_S^{(2)}$ has a transition that aggregates two transitions of $\mathcal{M}_S^{(1)}$.

Let $\pi^{(1)}$ be the steady state probability vector of $\mathcal{M}^{(1)}$ and $\pi^{(2)}$ be the steady state probability vector of $\mathcal{M}^{(2)}$.

Following the same reasoning as in the proof of lemma D.2, by construction, we can say that for every state $s_j \in S$, there exists $\psi_{s_0 \rightsquigarrow s_j}$ such that $\pi^{(1)}_{s_j} = \psi_{s_0 \rightsquigarrow s_j}\, \pi^{(1)}_{s_0}$ and $\pi^{(2)}_{s_j} = \psi_{s_0 \rightsquigarrow s_j}\, \pi^{(2)}_{s_0}$.

Because $\sum_{s' \in S \cup \{s\}} \pi^{(1)}_{s'} = 1$ and $\sum_{s' \in S} \pi^{(2)}_{s'} = 1$, we have that:

$$\pi^{(1)}_{s_0}\left(\sum_{s_j \in S} \psi_{s_0 \rightsquigarrow s_j}\right) + \pi^{(1)}_{s} \quad = \quad \pi^{(2)}_{s_0}\left(\sum_{s_j \in S} \psi_{s_0 \rightsquigarrow s_j}\right)$$

By construction, the distributions of the sojourn time in states of $S \backslash \{s_0\}$ are the same in $\mathcal{M}^{(1)}_S$ and in $\mathcal{M}^{(2)}_S$. For every state $s_j \in S \backslash \{s_0\}$, let E_{s_j} be the expected value of the distribution of the sojourn time in s_j (in the SMCs $\mathcal{M}^{(1)}_S$ or $\mathcal{M}^{(2)}_S$).

However, the distributions of the sojourn time in s_0 differ in $\mathcal{M}^{(1)}_S$ and in $\mathcal{M}^{(2)}_S$. Let $E^{(1)}_{s_0}$ be the expected value of the sojourn time in s_0 in the SMC $\mathcal{M}^{(1)}_S$, and $E^{(2)}_{s_0}$ be the expected value of the sojourn time in s_0 in the SMC $\mathcal{M}^{(2)}_S$. Consider the expected value $E_{s_0 \not\rightsquigarrow s}$ in the SMC $\mathcal{M}^{(1)}_S$, defined by:

$$E_{s_0 \not\rightsquigarrow s} = E\left[\sum_{\substack{a \in \mathcal{A},\\ p' \in]0,1]\\ s_j \neq s}} \left\{\left]p'\, \mathcal{D}^{(1)}\left(s_0 \overset{a\ \ p'}{\square\!\!\rightsquigarrow} s_j\right) \mid s_0 \overset{a\ \ p'}{\square\!\!\rightsquigarrow} s_j\right[\right\}\right]$$

$E_{s_0 \not\rightsquigarrow s}$ is the expected value of the distribution induced by transitions from s_0 but the transition $s_0 \overset{\tau\ \ p}{\square\!\!\rightsquigarrow} s$ (with $\mathcal{D}\left(s_0 \overset{\tau\ \ p}{\square\!\!\rightsquigarrow} s\right) = \mathbf{C}_n$).

By construction, For all transitions from s_0 different from $s_0 \overset{\tau\ \ p}{\square\!\!\rightsquigarrow} s$ in $\mathcal{M}^{(1)}_S$, there is the same transition with the same distribution in s_2. We can thus write that:

$$\begin{array}{rclcl} E^{(1)}_{s_0} & = & E_{s_0 \not\rightsquigarrow s} + p\, E\left[\mathbf{C}_n\right] & = & E_{s_0 \not\rightsquigarrow s} + p\, n \\ E^{(2)}_{s_0} & = & E_{s_0 \not\rightsquigarrow s} + p\, E\left[\mathbf{C}_{n+1}\right] & = & E_{s_0 \not\rightsquigarrow s} + p\, (n+1) \end{array}$$

In other words, we have $E^{(2)}_{s_0} = E^{(1)}_{s_0} + p$.

Finally, we can compute the steady state probability in the state s_0 of $\mathcal{M}_S^{(2)}$:

$$\begin{aligned}
\pi_{s_0}^{(2)'} &= \frac{\pi_{s_0}^{(2)}\, E_{s_0}^{(2)}}{\sum_{s_j \in S\setminus\{s_0\}} \left(\pi_{s_j}^{(2)}\, E_{s_j}\right) + \pi_{s_0}^{(2)}\, E_{s_0}^{(2)}} \\
&= \frac{\pi_{s_0}^{(2)}\, E_{s_0}^{(2)}}{\pi_{s_0}^{(2)}\left(\sum_{s_j \in S\setminus\{s_0\}} \left(\psi_{s_0 \rightsquigarrow s_j}\, E_{s_j}\right)\right) + \pi_{s_0}^{(2)}\, E_{s_0}^{(2)}} \\
&= \frac{E_{s_0}^{(2)}}{\sum_{s_j \in S\setminus\{s_0\}} \left(\psi_{s_0 \rightsquigarrow s_j}\, E_{s_j}\right) + E_{s_0}^{(2)}} \\
&= \frac{\pi_{s_0}^{(1)}\, E_{s_0}^{(2)}}{\pi_{s_0}^{(1)}\left(\sum_{s_j \in S\setminus\{s_0\}} \left(\psi_{s_0 \rightsquigarrow s_j}\, E_{s_j}\right)\right) + \pi_{s_0}^{(1)}\, E_{s_0}^{(2)}} \\
&= \frac{\pi_{s_0}^{(1)}\, E_{s_0}^{(1)} + p\, \pi_{s_0}^{(1)}}{\sum_{s_j \in S\setminus\{s_0\}} \left(\psi_{s_0 \rightsquigarrow s_j}\, \pi_{s_0}^{(1)}\, E_{s_j}\right) + \pi_{s_0}^{(1)}\, E_{s_0}^{(1)} + p\, \pi_{s_0}^{(1)}} \\
&= \frac{\pi_{s_0}^{(1)}\, E_{s_0}^{(1)} + \pi_s^{(1)}}{\sum_{s_j \in S\setminus\{s_0\}} \left(\pi_{s_j}^{(1)}\, E_{s_j}\right) + \pi_{s_0}^{(1)}\, E_{s_0}^{(1)} + \pi_s^{(1)}} \\
&= \pi_{s_0}^{(1)'} + \pi_s^{(1)'}
\end{aligned}$$

Similarly, we prove that for a state $s_j \neq s_0$, we have $\pi_{s_j}^{(2)'} = \pi_{s_j}^{(1)'}$ □

Finally, we can prove the following theorem, corresponding lemma 5.9.

Theorem D.1 (Steady state probabilities in the associated SMC)**.** *Consider a DTAMC $\mathcal{M}(P)$ associated to an IPC P, $\mathcal{M}(P) = \langle C, \rightsquigarrow, c_0, \mathscr{A}\rangle$, and $\mathcal{M}_S(P) = \langle S, \mathcal{A}, \square\!\!\rightsquigarrow, \mathcal{D}, s_0\rangle$ the SMC associated to $\mathcal{M}(P)$. Let k be the cardinal of the largest annotation multiset associated to states of $\mathcal{M}(P)$. We note π_c the steady state probability of a state $c \in C$ in $\mathcal{M}(P)$ and $\tilde{\pi}_s$ the steady state probability of a state $s \in S$ in $\mathcal{M}_S$.*

Consider a state $\langle s, \mathscr{S}\rangle \in C$ where $\mathscr{S}_2 = \{a_1, \ldots, a_n\}$ ($n \leq k$ and $\left(\forall i \leq n\right)\ a_i \in \mathcal{A}$), and $\mathscr{S}_{\rightsquigarrow\langle s,\mathscr{S}\rangle}$ the set of predecessors of $\langle s, \mathscr{S}\rangle$ in $\mathcal{M}(P)$. For every state $\langle s_i, \mathscr{S}_i\rangle \in \mathscr{S}_{\rightsquigarrow\langle s,\mathscr{S}\rangle}$ such that $\langle s_i, \mathscr{S}_i\rangle \overset{p_i}{\rightsquigarrow} \langle s, \mathscr{S}\rangle$, there is a sequence of transitions:

$$s_i \overset{\tau\ p_i}{\square\!\!\rightsquigarrow} s'_{i,1} \overset{a_1\ 1}{\square\!\!\rightsquigarrow} s'_{i,2} \overset{a_2\ 1}{\square\!\!\rightsquigarrow} \cdots \overset{a_{n-1}\ 1}{\square\!\!\rightsquigarrow} s'_{i,n} \overset{a_n\ 1}{\square\!\!\rightsquigarrow} s_2$$

in $\mathcal{M}_S$. Steady state probabilities of predecessors of s_2 in $\mathcal{M}_S$ are linked to the steady state probability of s_2 in $\mathcal{M}(P)$:

$$\left(\forall j < n\right) \quad \sum_{\langle s_i, \mathscr{S}_i\rangle \in \mathscr{S}_{\rightsquigarrow\langle s,\mathscr{S}\rangle}} \left(\tilde{\pi}_{s_{i,j}}\right) = \frac{\pi_{\langle s,\mathscr{S}\rangle}}{k}$$

Proof. This is a direct application of lemmas D.2, D.3, and D.4. The transformation from a DTAMC to an SMC can be seen as three successive transformations keeping strong links between the steady state probabilities of each transformation:

– firstly, the DTAMC is transformed in a k-scaled DTMC.
– secondly, to preserve the annotations, the k-scaled DTMC is transformed into an SMC such that the distribution associated to each transition is $\mathbf{C}_1$. The sequences of intermediate transitions added in the k-scaled DTMC are labeled with actions stored in states of the DTAMC or with τ.
– finally, transitions labeled with τ and probability one are aggregated (cf. lemma D.4)

□

Appendix E

LOTOS models

In this appendix we present the LOTOS models of push and pop queues, used in chapter 7. In those models, the peek operation is not implemented.

E.1 Data Type Libraries

E.1.1 Queue Size

```
type FifoSize is Boolean
   sorts FifoSize (*! implementedby ST_FIFOSIZE
                     comparedby ST_FIFOSIZE_CMP
                     iteratedby ST_FIFOSIZE_ENUM_FIRST
                            and ST_FIFOSIZE_ENUM_NEXT
                     printedby ST_FIFOSIZE_PRINT external *)
   opns
      0     (*! implementedby ST_FIFOSIZE_0 constructor external *),
      1     (*! implementedby ST_FIFOSIZE_1 constructor external *),
      2     (*! implementedby ST_FIFOSIZE_2 constructor external *),
      4     (*! implementedby ST_FIFOSIZE_4 constructor external *),
      8     (*! implementedby ST_FIFOSIZE_8 constructor external *),
      16    (*! implementedby ST_FIFOSIZE_16 constructor external *),
      32    (*! implementedby ST_FIFOSIZE_32 constructor external *),
      64    (*! implementedby ST_FIFOSIZE_64 constructor external *),
      128   (*! implementedby ST_FIFOSIZE_128 constructor external *),
      256   (*! implementedby ST_FIFOSIZE_256 constructor external *)
            : -> FifoSize

      _+_   (*! implementedby ST_FIFOSIZE_PLUS external *),
      _-_   (*! implementedby ST_FIFOSIZE_MINUS external *)
            : FifoSize, FifoSize -> FifoSize

      _==_ (*! implementedby ST_FIFOSIZE_EQ external *),
      _<>_ (*! implementedby ST_FIFOSIZE_NE external *),
      _<_  (*! implementedby ST_FIFOSIZE_LT external *),
      _<=_ (*! implementedby ST_FIFOSIZE_LE external *),
      _>_  (*! implementedby ST_FIFOSIZE_GT external *),
      _>=_ (*! implementedby ST_FIFOSIZE_GE external *)
```

```
          : FifoSize, FifoSize -> Bool
endtype
```

E.1.2 Queue Elements

```
type DataType is Boolean
   sorts DataType
   opns
      DATA (*! constructor *) : -> DataType
      SIGNAL (*! constructor *) : -> DataType
      _==_ ,
      _<>_ : DataType, DataType -> Bool

   eqns
      forall X,Y:DataType
      ofsort Bool
         DATA == DATA = true;
         SIGNAL == SIGNAL = true;
         DATA == SIGNAL = false;
         X == Y = Y == X;
     ofsort Bool
         X <> Y = not(X == Y);
endtype
```

E.1.3 Queue

```
type FIFO_TYPE is FifoSize, DataType
   sorts
      Fifo

   opns
      Empty (*! constructor *) : -> Fifo
      TPUSH (*! constructor *) : Fifo, DataType -> Fifo
      TPOP : Fifo -> Fifo
      HEAD : Fifo -> DataType
      SIZE : Fifo -> FifoSize

   eqns
      forall D:DataType, M:FifoSize, F:Fifo

         ofsort Fifo
            TPOP( TPUSH( Empty, D)) = Empty;
            TPOP( TPUSH( F, D)) = TPUSH( TPOP(F), D);

         ofsort DataType
            HEAD( TPUSH( Empty, D)) = D;
            HEAD( TPUSH( F, D)) = HEAD(F);

         ofsort FifoSize
            Size(Empty) = 0;
            Size( TPUSH( F, D)) = Size(F) + 1;

endtype
```

E.1.4 Push Queue Identifiers

```
type PUSHQ_ID is BOOLEAN
   sorts PUSHQ_ID

   opns
      PUSHQ_ID_1 (*! constructor *) : -> PUSHQ_ID
      PUSHQ_ID_2 (*! constructor *) : -> PUSHQ_ID
      _<>_ : PUSHQ_ID, PUSHQ_ID -> Bool
      _==_ : PUSHQ_ID, PUSHQ_ID -> Bool

   eqns
      forall q1,q2: PUSHQ_ID
         ofsort Bool
            q1 <> q1 = false;
            q1 <> q2 = true;
            q1 == q1 = true;
            q1 == q2 = false;
endtype
```

E.1.5 Pop Queue Identifiers

```
type POPQ_ID is BOOLEAN
   sorts POPQ_ID

   opns
      POPQ_ID_1 (*! constructor *) : -> POPQ_ID
      POPQ_ID_2 (*! constructor *) : -> POPQ_ID
      _<>_ : POPQ_ID, POPQ_ID -> Bool
      _==_ : POPQ_ID, POPQ_ID -> Bool

   eqns
      forall q1,q2: POPQ_ID
         ofsort Bool
            q1 <> q1 = false;
            q1 <> q2 = true;
            q1 == q1 = true;
            q1 == q2 = false;
endtype
```

E.2 Push Queue

```
(* list of used gates :
   PUSH_RQ      : request for a push operation
   PUSH_RSP     : response for a pop operation
   PUSH_DL_B    : beginning of the delay associated to the push operation
   PUSH_DL_E    : end of the delay associated to the push operation
   OUT_RQ       : request for the witdrawal operation
   OUT_RSP      : response for the witdrawal operation
   CDT_RQ       : request for the operation to receive a credit message
   CDT_RSP      : response for the operation to receive a credit message
   CDT_DL_B     : beginning of the delay associated to the credit operation
   CDT_DL_E     : end of the delay associated to the credit operation
```

```
   PUSHQ_ERROR : a gate of error to limit the size of the push queue model

   The implementation of the credit protocol implies that the model of the
   push queue is infinite. Only the association of a push queue model to a
   pop queue model leads to a finite model. Consequently, we use the
   PUSHQ_ERROR gate not to limit the size of the model.
*)

specification TOP [ PUSH_RQ, PUSH_RSP, PUSH_DL_B, PUSH_DL_E,
                    OUT_RQ , OUT_RSP ,
                    CDT_RQ , CDT_RSP , CDT_DL_B , CDT_DL_E,
                    PUSHQ_ERROR ] : noexit

library
  FIFO_SIZE, X_BOOLEAN, POPQ_ID, PUSHQ_ID, DATA_TYPE, FIFO_TYPE
endlib

behaviour

        PUSHQ [PUSH_RQ,  PUSH_RSP,  PUSH_DL_B,  PUSH_DL_E,
               OUT_RQ, OUT_RSP,
               CDT_RQ, CDT_RSP, CDT_DL_B, CDT_DL_E]
              (PUSHQ_ID_1 of PUSHQ_ID,
               POPQ_ID_1 of POPQ_ID,
               1 of FIFOSIZE,
               8+4+2 of FIFOSIZE)

        |[ CDT_RQ, CDT_RSP, OUT_RSP]|

        dummyEnv [CDT_RQ, CDT_RSP, OUT_RSP, PUSHQ_ERROR]
                 (PUSHQ_ID_1 of PUSHQ_ID,
                  8+4+2 of FifoSize)

where

(*^^^
```

```
        isPending(Pending) = true;
        isPending(S)       = false;

endtype

(*************************************************************************)

process dummyEnv [CDT_RQ, CDT_RSP, OUT_RSP, ERROR]
                 (id       : pushq_id,
                  popqsize : FifoSize) : noexit :=

   dummy_ctl [CDT_RQ, CDT_RSP, OUT_RSP, ERROR]
             (id,popqsize, popqsize+popqsize, 0 of FifoSize, popqsize,
              Idle of DUMMY_STATUS)
endproc

(* This process limits the value of the credit message that can be
   received by the push queue.
 *)
process dummy_ctl [CDT_RQ, CDT_RSP, OUT_RSP, ERROR]
                  ( id         : pushq_id, (* ID of the push queue *)
                    popqsize   : FifoSize, (* size of the pop queue *)
                    cdt_max    : FifoSize, (* max. value of credit in
                                              the push queue *)
                    last_rq    : FifoSize, (* last credit value sent *)
                    cur_credit : FifoSize, (* current credit of the
                                              push queue *)
                    status     : DUMMY_STATUS (* status for the credit *)
                  ) : noexit :=
   OUT_RSP;
      dummy_ctl [CDT_RQ, CDT_RSP, OUT_RSP, ERROR]
                (id,popqsize,cdt_max, last_rq, cur_credit-1, status)
   []
   [isIdle(status)] ->
   CDT_RQ !id ?N:FifoSize [(N>0) and (N<=popqsize)];
      dummy_ctl [CDT_RQ, CDT_RSP, OUT_RSP, ERROR]
                (id,popqsize,cdt_max, N, cur_credit,
                 Pending of DUMMY_STATUS)
   []
   [isPending(status)]->
   CDT_RSP;
   (
      let c:FIFOSIZE = cur_credit+last_rq in
      (
         [c > cdt_max] ->
            ERROR;
               stop
         []
         [c <= cdt_max] ->
            dummy_ctl [CDT_RQ, CDT_RSP, OUT_RSP, ERROR]
                      (id,popqsize,cdt_max, 0 of FifoSize, c,
                       Idle of DUMMY_STATUS)
```

```
            )
        )
    endproc

(*************************************************************************)

    type PUSHQ_IN_STATUS is BOOLEAN, DATATYPE, FIFOSIZE
        sorts
            PUSHQ_IN_STATUS

        opns
            Idle       (*! constructor *) :                  -> PUSHQ_IN_STATUS
            Pending    (*! constructor *) : DATATYPE        -> PUSHQ_IN_STATUS
            DlStarted  (*! constructor *) : DATATYPE        -> PUSHQ_IN_STATUS
            DlStopped  (*! constructor *) : DATATYPE        -> PUSHQ_IN_STATUS

            isIdle, isPending              : PUSHQ_IN_STATUS -> Bool
            isDlStarted, isDlStopped       : PUSHQ_IN_STATUS -> Bool
            getElm                         : PUSHQ_IN_STATUS -> DATATYPE

        eqns
            forall S:PUSHQ_IN_STATUS, X:DATATYPE, N:FIFOSIZE
            ofsort Bool
                isIdle(Idle) = true;
                isIdle(S)    = false;

                isPending(Pending(X)) = true;
                isPending(S)          = false;

                isDlStarted(DlStarted(X)) = true;
                isDlStarted(S)            = false;

                isDlStopped(DlStopped(X)) = true;
                isDlStopped(S)            = false;

            ofsort DATATYPE
                getElm(Pending(X))    = X;
                getElm(DlStarted(X)) = X;
                getElm(DlStopped(X)) = X;
    endtype

(*************************************************************************)

    type PUSHQ_OUT_STATUS is BOOLEAN, DATATYPE, FIFOSIZE
        sorts
            PUSHQ_OUT_STATUS

        opns
            Idle       (*! constructor *) :           -> PUSHQ_OUT_STATUS
            Pending    (*! constructor *) :           -> PUSHQ_OUT_STATUS
            DlStarted  (*! constructor *) :           -> PUSHQ_OUT_STATUS
            DlStopped  (*! constructor *) :           -> PUSHQ_OUT_STATUS
```

```
      isIdle , isPending              : PUSHQ_OUT_STATUS -> Bool
      isDlStarted , isDlStopped       : PUSHQ_OUT_STATUS -> Bool

   eqns
      forall S:PUSHQ_OUT_STATUS
      ofsort Bool
         isIdle(Idle) = true;
         isIdle(S)    = false;

         isPending(Pending) = true;
         isPending(S)       = false;

         isDlStarted(DlStarted) = true;
         isDlStarted(S)         = false;

         isDlStopped(DlStopped) = true;
         isDlStopped(S)         = false;
endtype

(*************************************************************************)

type PUSHQ_CDT_STATUS is BOOLEAN, DATATYPE, FIFOSIZE
   sorts
      PUSHQ_CDT_STATUS

   opns
      Idle       (*! constructor *) :           -> PUSHQ_CDT_STATUS
      Pending    (*! constructor *) : FIFOSIZE -> PUSHQ_CDT_STATUS
      DlStarted  (*! constructor *) : FIFOSIZE -> PUSHQ_CDT_STATUS
      DlStopped  (*! constructor *) : FIFOSIZE -> PUSHQ_CDT_STATUS

      isIdle , isPending              : PUSHQ_CDT_STATUS -> Bool
      isDlStarted , isDlStopped       : PUSHQ_CDT_STATUS -> Bool
      getCdt                          : PUSHQ_CDT_STATUS -> FIFOSIZE

   eqns
      forall S:PUSHQ_CDT_STATUS, D:DATATYPE, N:FIFOSIZE
      ofsort Bool
         isIdle(Idle) = true;
         isIdle(S)    = false;

         isPending(Pending(N)) = true;
         isPending(S)          = false;

         isDlStarted(DlStarted(N)) = true;
         isDlStarted(S)            = false;

         isDlStopped(DlStopped(N)) = true;
         isDlStopped(S)            = false;

      ofsort FIFOSIZE
         getCdt(Pending(N))    = N;
         getCdt(DlStarted(N)) = N;
```

```
            getCdt(DlStopped(N)) = N;
   endtype

(*********************************************************************)

process PUSHQ [IN_RQ, IN_RSP, IN_DL_B, IN_DL_E,
               OUT_RQ, OUT_RSP,
               CDT_RQ, CDT_RSP, CDT_DL_B, CDT_DL_E]
              ( source         : pushq_id, (* ID of the push queue *)
                target         : popq_id,  (* ID of the targeted pop queue *)
                SMax           : FifoSize, (* size of the push queue *)
                SMaxPopQ: FifoSize         (* size of the targeted pop queue *)
              ) : noexit :=

   PUSHQ_ctl  [IN_RQ, IN_RSP, IN_DL_B, IN_DL_E,
               OUT_RQ , OUT_RSP ,
               CDT_RQ , CDT_RSP , CDT_DL_B , CDT_DL_E]
              (source, target, SMax, Empty of Fifo, 0 of FifoSize,
               Idle of PUSHQ_IN_STATUS,
               Idle of PUSHQ_CDT_STATUS,
               Idle of PUSHQ_OUT_STATUS, SMaxPopQ)

where

   process PUSHQ_ctl [IN_RQ, IN_RSP, IN_DL_B, IN_DL_E,
                      OUT_RQ, OUT_RSP,
                      CDT_RQ, CDT_RSP, CDT_DL_B, CDT_DL_E]
                     ( my_id      : pushq_id, (* ID of the push queue *)
                       dst_id     : popq_id,  (* ID of the target pop queue *)
                       SMax       : FifoSize, (* size of the push queue *)
                       F          : FIFO,     (* queue storing elements *)
                       SCur       : FifoSize, (* current number of elements *)
                       inStatus   : PUSHQ_IN_STATUS,  (* status for input *)
                       cdtStatus  : PUSHQ_CDT_STATUS, (* status for credit *)
                       outStatus  : PUSHQ_OUT_STATUS, (* status for output *)
                       credit     : FifoSize          (* current credit *)
                     ) : noexit :=

      (* management of the PUSH part of the behavior *)
      [isIdle(inStatus)] ->
         IN_RQ ?X:DATATYPE;
            PUSHQ_ctl [IN_RQ, IN_RSP, IN_DL_B, IN_DL_E,
                       OUT_RQ, OUT_RSP,
                       CDT_RQ, CDT_RSP, CDT_DL_B, CDT_DL_E]
                      ( my_id, dst_id, SMax, F, SCur,
                        Pending(X) of PUSHQ_IN_STATUS,
                        cdtStatus, outStatus, credit)
      []
      [isPending(inStatus) and (SCur<SMax)] ->
         IN_DL_B;
            PUSHQ_ctl [IN_RQ, IN_RSP, IN_DL_B, IN_DL_E,
                       OUT_RQ, OUT_RSP,
                       CDT_RQ, CDT_RSP, CDT_DL_B, CDT_DL_E]
```

```
                    ( my_id, dst_id, SMax, F, SCur,
                      DlStarted(getElm(inStatus)) of PUSHQ_IN_STATUS,
                      cdtStatus, outStatus, credit)
    []
    [isDlStarted(inStatus)] ->
       IN_DL_E;
          PUSHQ_ctl [IN_RQ, IN_RSP, IN_DL_B, IN_DL_E,
                     OUT_RQ, OUT_RSP,
                     CDT_RQ, CDT_RSP, CDT_DL_B, CDT_DL_E]
                    ( my_id, dst_id, SMax, F, SCur,
                      DlStopped(getElm(inStatus)) of PUSHQ_IN_STATUS,
                      cdtStatus, outStatus, credit)
    []
    [isDlStopped(inStatus)] ->
       IN_RSP;
          PUSHQ_ctl [IN_RQ, IN_RSP, IN_DL_B, IN_DL_E,
                     OUT_RQ, OUT_RSP,
                     CDT_RQ, CDT_RSP, CDT_DL_B, CDT_DL_E]
                    ( my_id, dst_id, SMax, TPUSH(F, getElm(inStatus)),
                      SCur+1, Idle of PUSHQ_IN_STATUS,
                      cdtStatus, outStatus, credit)
    []

    (* management of the "send to NOC" part of the behavior *)
    [isIdle(outStatus) and (SCur > 0) and (credit > 0)] ->
       OUT_RQ !dst_id !Head(F);
          (* To keep equivalence with simple queue,
             POP cannot be done here *)
          PUSHQ_ctl [IN_RQ, IN_RSP, IN_DL_B, IN_DL_E,
                     OUT_RQ, OUT_RSP,
                     CDT_RQ, CDT_RSP, CDT_DL_B, CDT_DL_E]
                    ( my_id, dst_id, SMax, F, SCur,
                      inStatus, cdtStatus, Pending of PUSHQ_OUT_STATUS,
                      credit)
    []
    [isPending(outStatus)] ->
       OUT_RSP;
          PUSHQ_ctl [IN_RQ, IN_RSP, IN_DL_B, IN_DL_E,
                     OUT_RQ, OUT_RSP,
                     CDT_RQ, CDT_RSP, CDT_DL_B, CDT_DL_E]
                    ( my_id, dst_id, SMax, TPOP(F), SCur-1,
                      inStatus, cdtStatus,
                      Idle of PUSHQ_OUT_STATUS, credit-1)
    []

    (* management of the "response to credit" of the behavior *)
    [isIdle(cdtStatus)] ->
       CDT_RQ !my_id ?c:FifoSize;
          PUSHQ_ctl [IN_RQ, IN_RSP, IN_DL_B, IN_DL_E,
                     OUT_RQ, OUT_RSP,
                     CDT_RQ, CDT_RSP, CDT_DL_B, CDT_DL_E]
                    ( my_id, dst_id, SMax, F, SCur,
                      inStatus, Pending(c) of PUSHQ_CDT_STATUS,
```

```
                        outStatus, credit)
      []
      [isPending(cdtStatus)] ->
         CDT_DL_B;
            PUSHQ_ctl [IN_RQ, IN_RSP, IN_DL_B, IN_DL_E,
                       OUT_RQ, OUT_RSP,
                       CDT_RQ, CDT_RSP, CDT_DL_B, CDT_DL_E]
                      ( my_id, dst_id, SMax, F, SCur, inStatus,
                        DlStarted(getCdt(cdtStatus)) of PUSHQ_CDT_STATUS,
                        outStatus, credit)
      []
      [isDlStarted(cdtStatus)] ->
         CDT_DL_E;
            PUSHQ_ctl [IN_RQ, IN_RSP, IN_DL_B, IN_DL_E,
                       OUT_RQ, OUT_RSP,
                       CDT_RQ, CDT_RSP, CDT_DL_B, CDT_DL_E]
                      ( my_id, dst_id, SMax, F, SCur, inStatus,
                        DlStopped(getCdt(cdtStatus)) of PUSHQ_CDT_STATUS,
                        outStatus, credit)
      []
      [isDlStopped(cdtStatus)] ->
         CDT_RSP;
            PUSHQ_ctl [IN_RQ, IN_RSP, IN_DL_B, IN_DL_E,
                       OUT_RQ, OUT_RSP,
                       CDT_RQ, CDT_RSP, CDT_DL_B, CDT_DL_E]
                      ( my_id, dst_id, SMax, F, SCur, inStatus,
                        Idle of PUSHQ_CDT_STATUS,
                        outStatus, credit+getCdt(cdtStatus))
   endproc

endproc

endspec
```

E.3 Pop Queue

```
(* list of used gates :
   IN_RQ      : request for an insertion operation
   IN_RSP     : response for an insertion operation
   IN_DL_B    : beginning of the delay associated to the insertion operation
   IN_DL_E    : end of the delay associated to the insertion operation
   POP_RQ     : request for a pop operation
   POP_RSP    : response for a pop operation
   POP_DL_B   : beginning of the delay associated to the pop operation
   POP_DL_E   : end of the delay associated to the pop operation
   CDT_RQ     : request for the operation to send a credit message
   CDT_RSP    : response for the operation to send a credit message

   The implementation of the credit protocol implies that the model of the
   push queue is infinite. Only the association of a push queue model to a
   pop queue model leads to a finite model. Consequently, we use the
   PUSHQ_ERROR gate not to limit the size of the model.
*)
```

```
specification TOP_POPQ [ IN_RQ,  IN_RSP,  IN_DL_B,  IN_DL_E,
                         POP_RQ, POP_RSP, POP_DL_B, POP_DL_E,
                         CDT_RQ, CDT_RSP] : noexit

library
 FIFO_SIZE, X_BOOLEAN, PUSHQ_ID, POPQ_ID, DATA_TYPE, FIFO_TYPE
endlib

behaviour

   POPQ [IN_RQ,  IN_RSP,  IN_DL_B,  IN_DL_E,
         POP_RQ, POP_RSP, POP_DL_B, POP_DL_E,
         CDT_RQ, CDT_RSP]

        (PUSHQ_ID_1 of PUSHQ_ID,
         POPQ_ID_1 of POPQ_ID,
         8+4+2 of FIFOSIZE,
         1 of FIFOSIZE)

   |[IN_RQ, CDT_RQ]|

   dummyEnv [IN_RQ, CDT_RQ]
            (PUSHQ_ID_1 of PUSHQ_ID,
             POPQ_ID_1 of POPQ_ID,
             8+4+2 of FIFOSIZE)

where

(*************************************************************************)

   process dummyEnv [IN_RQ, CDT_RQ]
                    ( id_src   : pushq_id,
                      id       : popq_id,
                      popqsize : FifoSize) : noexit :=

      dummy_ctl [IN_RQ, CDT_RQ] (id_src, id, popqsize + popqsize)

   endproc

   process dummy_ctl [IN_RQ, CDT_RQ]
                     ( id_src   : pushq_id,
                       id       : popq_id,
                       credit   : FifoSize) : noexit :=
      [credit > 0 ] ->
      IN_RQ !id ?X:DataType;
         dummy_ctl [IN_RQ, CDT_RQ] (id_src, id, credit - 1)
      []
      CDT_RQ !id_src ?C:FifoSize;
         dummy_ctl [IN_RQ, CDT_RQ] (id_src, id, credit + C)
   endproc

(*************************************************************************)
```

```
type POPQ_IN_STATUS is BOOLEAN, FIFOSIZE, DATATYPE
   sorts
      POPQ_IN_STATUS

   opns
      Idle          (*! constructor *) :                  -> POPQ_IN_STATUS
      Pending       (*! constructor *) : DATATYPE         -> POPQ_IN_STATUS
      DlStarted     (*! constructor *) : DATATYPE         -> POPQ_IN_STATUS
      DlStopped     (*! constructor *) : DATATYPE         -> POPQ_IN_STATUS

      isIdle, isPending                : POPQ_IN_STATUS -> Bool
      isDlStarted, isDlStopped         : POPQ_IN_STATUS -> Bool
      getElm                           : POPQ_IN_STATUS -> DATATYPE

   eqns
      forall S:POPQ_IN_STATUS, X:DATATYPE
      ofsort Bool
         isIdle(Idle) = true;
         isIdle(S)    = false;

         isPending(Pending(X))  = true;
         isPending(S)           = false;

         isDlStarted(DlStarted(X)) = true;
         isDlStarted(S)            = false;

         isDlStopped(DlStopped(X)) = true;
         isDlStopped(S)            = false;

      ofsort DATATYPE
         getElm(Pending(X))    = X;
         getElm(DlStarted(X)) = X;
         getElm(DlStopped(X)) = X;
endtype

(*-------------------------------------------------------------------*)

type POPQ_OUT_STATUS is BOOLEAN, FIFOSIZE, DATATYPE
   sorts
      POPQ_OUT_STATUS

   opns
      Idle          (*! constructor *) :                  -> POPQ_OUT_STATUS
      Pending       (*! constructor *) :                  -> POPQ_OUT_STATUS
      DlStarted     (*! constructor *) :                  -> POPQ_OUT_STATUS
      DlStopped     (*! constructor *) :                  -> POPQ_OUT_STATUS

      isIdle, isPending                : POPQ_OUT_STATUS -> Bool
      isDlStarted, isDlStopped         : POPQ_OUT_STATUS -> Bool

   eqns
      forall S:POPQ_OUT_STATUS
```

```
        ofsort Bool
           isIdle(Idle)           = true;
           isIdle(S)              = false;

           isPending(Pending)     = true;
           isPending(S)           = false;

           isDlStarted(DlStarted) = true;
           isDlStarted(S)         = false;

           isDlStopped(DlStopped) = true;
           isDlStopped(S)         = false;
   endtype

(*************************************************************************)

   type POPQ_CDT_STATUS is BOOLEAN, FIFOSIZE
      sorts
         POPQ_CDT_STATUS

      opns
         Idle      (*! constructor *) :           -> POPQ_CDT_STATUS
         Pending   (*! constructor *) : FIFOSIZE -> POPQ_CDT_STATUS

         isIdle, isPending : POPQ_CDT_STATUS -> Bool
        getCdt : POPQ_CDT_STATUS -> FifoSize

      eqns
         forall S:POPQ_CDT_STATUS, N:FIFOSIZE
         ofsort Bool
            isIdle(Idle) = true;
            isIdle(S)    = false;

            isPending(Pending(N)) = true;
            isPending(S)          = false;

         ofsort FIFOSIZE
            getCdt(Pending(N)) = N;
   endtype

(*************************************************************************)

process POPQ [IN_RQ, IN_RSP, IN_DL_B, IN_DL_E,
              OUT_RQ, OUT_RSP, OUT_DL_B, OUT_DL_E,
              CDT_RQ, CDT_RSP]
             ( source    : pushq_id, (* ID of the source push queue *)
               target    : popq_id,  (* ID of the pop queue *)
               SMax      : FifoSize, (* size of the pop queue *)
               Threshold : FifoSize  (* threshold for credit *)
             ) : noexit :=

      POPQ_ctl [IN_RQ, IN_RSP, IN_DL_B, IN_DL_E,
                OUT_RQ, OUT_RSP, OUT_DL_B, OUT_DL_E,
```

```
                CDT_RQ, CDT_RSP]
               (source, target, SMax, Threshold, Empty of Fifo,
                0 of FIFOSIZE, 0 of FIFOSIZE,
                Idle of POPQ_IN_STATUS, Idle of POPQ_OUT_STATUS,
                Idle of POPQ_CDT_STATUS)

where

   process POPQ_ctl [IN_RQ, IN_RSP, IN_DL_B, IN_DL_E,
                     OUT_RQ, OUT_RSP, OUT_DL_B, OUT_DL_E,
                     CDT_RQ, CDT_RSP]
                    ( src_id     : pushq_id,
                      my_id      : popq_id,
                      SMax       : FifoSize,
                      Threshold  : FifoSize,
                      F          : Fifo,
                      SCur       : FifoSize,
                      Credit     : FifoSize,
                      inStatus   : POPQ_IN_STATUS,
                      outStatus  : POPQ_OUT_STATUS,
                      cdtStatus  : POPQ_CDT_STATUS) : noexit :=

   (* management of the out of queue (pop) part of the behavior *)
   (* ------------------------------------------------------------------ *)

      (* we can accept a POP_RQ *)
      [isIdle(outStatus)] ->
         OUT_RQ;
            POPQ_ctl [IN_RQ, IN_RSP, IN_DL_B, IN_DL_E,
                      OUT_RQ, OUT_RSP, OUT_DL_B, OUT_DL_E,
                      CDT_RQ, CDT_RSP]
                     ( src_id, my_id, SMax, Threshold, F, SCur, Credit,
                       inStatus, Pending, cdtStatus)
      []
      (* the  POP delay can be started iff the queue is not empty *)
      [isPending(outStatus) and (SCur>0)] ->
         OUT_DL_B;
            POPQ_ctl [IN_RQ, IN_RSP, IN_DL_B, IN_DL_E,
                      OUT_RQ, OUT_RSP, OUT_DL_B, OUT_DL_E,
                      CDT_RQ, CDT_RSP]
                     ( src_id, my_id, SMax, Threshold, F, SCur, Credit,
                       inStatus, DlStarted, cdtStatus)
      []
      (* the pop delay is over after having been started *)
      [isDlStarted(outStatus)] ->
         OUT_DL_E;
            POPQ_ctl [IN_RQ, IN_RSP, IN_DL_B, IN_DL_E,
                      OUT_RQ, OUT_RSP, OUT_DL_B, OUT_DL_E,
                      CDT_RQ, CDT_RSP]
                     ( src_id, my_id,  SMax, Threshold, F, SCur, Credit,
                       inStatus, DlStopped, cdtStatus)
      []
      (* the pop delay is over, we grant the pop *)
```

```
    [isDlStopped(outStatus)] ->
       OUT_RSP !HEAD(F);
          POPQ_ctl [IN_RQ, IN_RSP, IN_DL_B, IN_DL_E,
                    OUT_RQ, OUT_RSP, OUT_DL_B, OUT_DL_E,
                    CDT_RQ, CDT_RSP]
                   ( src_id , my_id, SMax, Threshold, TPOP(F),
                     SCur-1, Credit+1, inStatus,
                     Idle of POPQ_OUT_STATUS, cdtStatus)
    []

  (* management of the in of queue (push from noc) part of the behavior *)
  (* ------------------------------------------------------------------ *)
    [isIdle(inStatus)] ->
       IN_RQ !my_id ?X:DataType;
          POPQ_ctl [IN_RQ, IN_RSP, IN_DL_B, IN_DL_E,
                    OUT_RQ, OUT_RSP, OUT_DL_B, OUT_DL_E,
                    CDT_RQ, CDT_RSP]
                   ( src_id , my_id, SMax, Threshold, F, SCur, Credit,
                     Pending(X), outStatus,cdtStatus)
    []
    [isPending(inStatus) and (SCur<SMax)] ->
       IN_DL_B;
          POPQ_ctl [IN_RQ, IN_RSP, IN_DL_B, IN_DL_E,
                    OUT_RQ, OUT_RSP, OUT_DL_B, OUT_DL_E,
                    CDT_RQ, CDT_RSP]
                   ( src_id , my_id, SMax, Threshold, F, SCur, Credit,
                     DlStarted(getElm(inStatus)), outStatus, cdtStatus)
    []
    [isDlStarted(inStatus)] ->
       IN_DL_E;
          POPQ_ctl [IN_RQ, IN_RSP, IN_DL_B, IN_DL_E,
                    OUT_RQ, OUT_RSP, OUT_DL_B, OUT_DL_E,
                    CDT_RQ, CDT_RSP]
                   ( src_id , my_id, SMax, Threshold, F, SCur, Credit,
                     DlStopped(getElm(inStatus)), outStatus, cdtStatus)
    []
    [isDlStopped(inStatus)] ->
       IN_RSP;
          POPQ_ctl [IN_RQ, IN_RSP, IN_DL_B, IN_DL_E,
                    OUT_RQ, OUT_RSP, OUT_DL_B, OUT_DL_E,
                    CDT_RQ, CDT_RSP]
                   ( src_id , my_id, SMax, Threshold,
                     TPUSH(F,getElm(inStatus)), SCur+1, Credit,
                     Idle of POPQ_IN_STATUS, outStatus, cdtStatus)
    []

  (* management of the credit part of the behavior *)
  (* --------------------------------------------- *)
    (* credit is greater than the threshold *)
    [(credit >= threshold) and isIdle(cdtStatus)] ->
       CDT_RQ !src_id !threshold;
          POPQ_ctl [IN_RQ, IN_RSP, IN_DL_B, IN_DL_E,
                    OUT_RQ, OUT_RSP, OUT_DL_B, OUT_DL_E,
```

```
                        CDT_RQ, CDT_RSP]
                       ( src_id , my_id, SMax, Threshold, F, SCur, Credit,
                         inStatus , outStatus , Pending(threshold))
      []
      [isPending(cdtStatus)] ->
         CDT_RSP;
            POPQ_ctl [IN_RQ, IN_RSP, IN_DL_B, IN_DL_E,
                      OUT_RQ, OUT_RSP, OUT_DL_B, OUT_DL_E,
                      CDT_RQ, CDT_RSP]
                     ( src_id , my_id, SMax, Threshold, F, SCur,
                       Credit-getCdt(CdtStatus),
                       instatus , outStatus , Idle of POPQ_CDT_STATUS)
   endproc

endproc

endspec
```

E.4 Network-on-Chip abstraction

```
(* list of used gates :
   IN_RQ      : request for an insertion operation
   IN_RSP     : response for an insertion operation
   IN_DL1_B   : beginning of the delay of the 1st one-place queue
   IN_DL1_E   : end of the delay of the 1st one-place queue
   IN_DL1_B   : beginning of the delay of the 2nd one-place queue
   IN_DL1_E   : end of the delay of the 2nd one-place queue
   IN_DL1_B   : beginning of the delay of the 3rd one-place queue
   IN_DL1_E   : end of the delay of the 3rd one-place queue
   OUT_RQ     : request for a pop operation
   OUT_RSP    : response for a pop operation

   we use a simple model of a FIFO queue (with blocking request/response
   operations) to abstract from the NoC.
*)

specification TOP [ IN_RQ, IN_RSP,
                    IN_DL1_B, IN_DL1_E,
                    IN_DL2_B, IN_DL2_E,
                    IN_DL3_B, IN_DL3_E,
                    OUT_RQ, OUT_RSP ] : noexit

library
 FIFO_SIZE, X_BOOLEAN, PUSHQ_ID, POPQ_ID, DATA_TYPE
endlib

behaviour

         NOC [IN_RQ, IN_RSP,
              IN_DL1_B, IN_DL1_E,
              IN_DL2_B, IN_DL2_E,
              IN_DL3_B, IN_DL3_E,
              OUT_RQ, OUT_RSP]
```

```
where

(*************************************************************************)
type NOC_ITEM is DataType, POPQ_ID
   sorts NOC_ITEM

   opns
      MAKE_ITEM (*! constructor *) : POPQ_ID, DATATYPE -> NOC_ITEM
      GET_DATA                     : NOC_ITEM -> DATATYPE
      GET_TARGET                   : NOC_ITEM -> POPQ_ID

   eqns
      forall D:DATATYPE, T:POPQ_ID
         ofsort DATATYPE
            GET_DATA( MAKE_ITEM( T, D)) = D;

         ofsort POPQ_ID
            GET_TARGET( MAKE_ITEM( T, D)) = T;
endtype

(*************************************************************************)
type NOC_IN_STATUS is BOOLEAN, NOC_ITEM
   sorts NOC_IN_STATUS

   opns
      Idle       (*! constructor *) :             -> NOC_IN_STATUS
      Pending    (*! constructor *) : NOC_ITEM -> NOC_IN_STATUS
      DlStarted  (*! constructor *) : NOC_ITEM -> NOC_IN_STATUS
      DlStopped  (*! constructor *) : NOC_ITEM -> NOC_IN_STATUS

      isIdle, isPending :         NOC_IN_STATUS -> Bool
      isDlStarted, isDlStopped : NOC_IN_STATUS -> Bool
      get_Item :                  NOC_IN_STATUS -> NOC_ITEM

   eqns
      forall S:NOC_IN_STATUS, N:NOC_ITEM
      ofsort Bool
         isIdle(Idle) = true;
         isIdle(S)    = false;

         isPending(Pending(N)) = true;
         isPending(S)          = false;

         isDlStarted(DlStarted(N)) = true;
         isDlStarted(S)            = false;

         isDlStopped(DlStopped(N)) = true;
         isDlStopped(S)            =false;

      ofsort NOC_ITEM
         get_Item(Pending(N))    = N;
         get_Item(DlStarted(N)) = N;
         get_Item(DlStopped(N)) = N;
```

```
endtype

(*************************************************************************)
type NOC_OUT_STATUS is BOOLEAN, DATATYPE, FIFOSIZE
   sorts
      NOC_OUT_STATUS

   opns
      Idle       (*! constructor *) :                  -> NOC_OUT_STATUS
      Pending    (*! constructor *) :                  -> NOC_OUT_STATUS

      isIdle, isPending             : NOC_OUT_STATUS -> Bool
   eqns
      forall S:NOC_OUT_STATUS
      ofsort Bool
         isIdle(Idle)       = true;
         isIdle(S)          = false;
         isPending(Pending) = true;
         isPending(S)       = false;
endtype

(*************************************************************************)
type NOC_FIFO is NOC_ITEM
   sorts NOC_FIFO

   opns
      Empty (*! constructor *) :                       -> NOC_FIFO
      TPUSH (*! constructor *) : NOC_FIFO, NOC_ITEM -> NOC_FIFO
      TPOP                     : NOC_FIFO             -> NOC_FIFO
      HEAD                     : NOC_FIFO             -> NOC_ITEM
   eqns
      forall N:NOC_ITEM, F:NOC_FIFO
         ofsort  NOC_FIFO
            TPOP( TPUSH( Empty, N)) = Empty;
            TPOP( TPUSH( F, N)) = TPUSH( TPOP(F), N);
            (* Due to priority in expressions,
               will not cover TPUSH( Empty, N) *)

         ofsort NOC_ITEM
            HEAD( TPUSH( Empty, N)) = N;
            HEAD( TPUSH( F, N)) = HEAD(F);
            (* Due to priority in expressions,
               will not cover TPUSH( Empty, N) *)
endtype

(*************************************************************************)
process NOC [IN_RQ, IN_RSP,
             IN_DL1_B, IN_DL1_E,
             IN_DL2_B, IN_DL2_E,
             IN_DL3_B, IN_DL3_E,
             OUT_RQ, OUT_RSP]
   hide OUT_B1_RQ, OUT_B1_RSP, OUT_B2_RQ, OUT_B2_RSP in
```

```
(
   NOC_ctl  [IN_RQ, IN_RSP,
             IN_DL1_B, IN_DL1_E,
             OUT_B1_RQ, OUT_B1_RSP]
            ( 1 of FifoSize, Empty of NOC_FIFO, 0 of FifoSize,
              Idle of NOC_IN_STATUS,
              Idle of NOC_OUT_STATUS)
   |[OUT_B1_RQ, OUT_B1_RSP]|
   NOC_ctl  [OUT_B1_RQ, OUT_B1_RSP,
             IN_DL2_B, IN_DL2_E,
             OUT_B2_RQ, OUT_B2_RSP]
            ( 1 of FifoSize, Empty of NOC_FIFO, 0 of FifoSize,
              Idle of NOC_IN_STATUS,
              Idle of NOC_OUT_STATUS)
   |[OUT_B2_RQ, OUT_B2_RSP]|
   NOC_ctl [OUT_B2_RQ, OUT_B2_RSP,
            IN_DL3_B, IN_DL3_E,
            OUT_RQ, OUT_RSP ]
           ( 1 of FifoSize, Empty of NOC_FIFO, 0 of FifoSize,
             Idle of NOC_IN_STATUS,
             Idle of NOC_OUT_STATUS)
)

where

   process NOC_ctl [IN_RQ, IN_RSP, IN_DL_B, IN_DL_E, OUT_RQ, OUT_RSP]
                   ( SMax      : FifoSize, (* size of the noc queue *)
                     F         : NOC_FIFO, (* queue storing elements *)
                     SCur      : FifoSize, (* current number of elements *)
                     inStatus  : NOC_IN_STATUS,  (* status for input *)
                     outStatus : NOC_OUT_STATUS  (* status for output *)
                   ) : noexit :=

      (* management of the PUSH part of the behavior *)
      [isIdle(inStatus)] ->
         IN_RQ ?Q:Popq_Id ?D:DataType;
            NOC_ctl [IN_RQ, IN_RSP, IN_DL_B, IN_DL_E, OUT_RQ, OUT_RSP]
                    ( SMax, F, Scur,
                      Pending(MAKE_ITEM(Q,D)) of NOC_IN_STATUS,
                      outStatus)
      []
      [isPending(inStatus) and (SCur<SMax)] ->
         IN_DL_B;
            NOC_ctl [IN_RQ, IN_RSP, IN_DL_B, IN_DL_E, OUT_RQ, OUT_RSP]
                    ( SMax, F, SCur,
                      DlStarted(get_Item(inStatus)) of NOC_IN_STATUS,
                      outStatus)
      []
      [isDlStarted(inStatus)] ->
         IN_DL_E;
            NOC_ctl [IN_RQ, IN_RSP, IN_DL_B, IN_DL_E, OUT_RQ, OUT_RSP]
                    ( SMax, F, SCur,
                      DlStopped(get_Item(inStatus)) of NOC_IN_STATUS,
```

```
                    outStatus)
      []
      [isDlStopped(inStatus)] ->
         IN_RSP;
            NOC_ctl [IN_RQ, IN_RSP, IN_DL_B, IN_DL_E, OUT_RQ, OUT_RSP]
                    ( SMax, TPUSH(F, get_Item(inStatus)), SCur+1,
                      Idle of NOC_IN_STATUS,
                      outStatus)
      []
      (* management of the POP part of the behavior *)
      [isIdle(outStatus) and (SCur > 0)] ->
         OUT_RQ !GET_TARGET(Head(F)) !GET_DATA(Head(F));
            NOC_ctl [IN_RQ, IN_RSP, IN_DL_B, IN_DL_E, OUT_RQ, OUT_RSP]
                    ( SMax, F, SCur, inStatus,
                      Pending of NOC_OUT_STATUS)
      []
      [isPending(outStatus)] ->
         OUT_RSP;
            NOC_ctl [IN_RQ, IN_RSP, IN_DL_B, IN_DL_E, OUT_RQ, OUT_RSP]
                    ( SMax, TPOP(F), SCur-1, inStatus,
                      Idle of NOC_OUT_STATUS)
   endproc
endproc

endspec
```

Bibliography

[ABDW06] Suzana Andova, Jos C. M. Baeten, Pedro R. D'Argenio, and Tim A. C. Willemse. A compositional merge of probabilistic processes in the alternating model, 2006.

[ACD93] Rajeev Alur, Costas Courcoubetis, and David L. Dill. Model-checking in dense real-time. *Inf. Comput.*, 104(1):2–34, 1993.

[AD91] Rajeev Alur and David L. Dill. The theory of timed automata. In J. W. de Bakker, Cornelis Huizing, Willem P. de Roever, and Grzegorz Rozenberg, editors, *REX Workshop*, volume 600 of *Lecture Notes in Computer Science*, pages 45–73. Springer, 1991.

[AD94] Rajeev Alur and David L. Dill. A theory of timed automata. *Theor. Comput. Sci.*, 126(2):183–235, 1994.

[AFH91] Rajeev Alur, Tomás Feder, and Thomas A. Henzinger. The benefits of relaxing punctuality. In *PODC*, pages 139–152, 1991.

[And00] Suzana Andova. Time and probability in process algebra. In Teodor Rus, editor, *AMAST*, volume 1816 of *Lecture Notes in Computer Science*, pages 323–338. Springer, 2000.

[ASB95] Adnan Aziz, Vigyan Singhal, and Felice Balarin. It usually works: The temporal logic of stochastic systems. In Pierre Wolper, editor, *CAV*, volume 939 of *Lecture Notes in Computer Science*, pages 155–165. Springer, 1995.

[AW06] Suzana Andova and Tim A. C. Willemse. Branching bisimulation for probabilistic systems: Characteristics and decidability. *Theor. Comput. Sci.*, 356(3):325–355, 2006.

[BA03] Mario Bravetti and Alessandro Aldini. Discrete time generative-reactive probabilistic processes with different advancing speeds. *Theor. Comput. Sci.*, 290(1):355–406, 2003.

[Bas96] Twan Basten. Branching bisimilarity is an equivalence indeed! *Inf. Process. Lett.*, 58(3):141–147, 1996.

[BB87] Tommaso Bolognesi and Ed Brinksma. Introduction to the iso specification language lotos. *Computer Networks*, 14:25–59, 1987.

[BB91] Jos C. M. Baeten and Jan A. Bergstra. Real time process algebra. *Formal Asp. Comput.*, 3(2):142–188, 1991.

[BB96] Jos C. M. Baeten and Jan A. Bergstra. Discrete time process algebra. *Formal Asp. Comput.*, 8(2):188–208, 1996.

[BDG+95] Marco Bernardo, Lorenzo Donatiello, Roberto Gorrieri, Piazza Porta, and S. Donato. Integrating performance and functional analysis of concurrent systems with empa. Technical report, University of Bologna, 1995.

[BDL+06] Gerd Behrmann, Alexandre David, Kim Guldstrand Larsen, John Håkansson, Paul Pettersson, Wang Yi, and Martijn Hendriks. Uppaal 4.0. In *QEST*, pages 125–126. IEEE Computer Society, 2006.

[BDM+98] Marius Bozga, Conrado Daws, Oded Maler, Alfredo Olivero, Stavros Tripakis, and Sergio Yovine. Kronos: A model-checking tool for real-time systems. In Alan J. Hu and Moshe Y. Vardi, editors, *CAV*, volume 1427 of *Lecture Notes in Computer Science*, pages 546–550. Springer, 1998.

[BdS92] Amar Bouali and Robert de Simone. Symbolic bisimulation minimisation. In Gregor von Bochmann and David K. Probst, editors, *CAV*, volume 663 of *Lecture Notes in Computer Science*, pages 96–108. Springer, 1992.

[Ber05] Gérard Berry. Esterel v7: From verified formal specification to efficient industrial designs. In Maura Cerioli, editor, *FASE*, volume 3442 of *Lecture Notes in Computer Science*, page 1. Springer, 2005.

[BFP01] Jan Bergstra, Wan Fokkink, and Alban Ponse. Process algebra with recursive operations. In *Handbook of Process Algebra*, pages 333–389. Elsevier, 2001.

[BH97] Christel Baier and Holger Hermanns. Weak bisimulation for fully probabilistic processes. In Orna Grumberg, editor, *CAV*, volume 1254 of *Lecture Notes in Computer Science*, pages 119–130. Springer, 1997.

[BHHK03] Christel Baier, Boudewijn R. Haverkort, Holger Hermanns, and Joost-Pieter Katoen. Model-checking algorithms for continuous-time markov chains. *IEEE Trans. Software Eng.*, 29(6):524–541, 2003.

[Bil95] Patrick Billingsley. *Probability and Measure*. Wiley, 1995.

[BK90] Jos C. M. Baeten and Jan Willem Klop, editors. *CONCUR '90, Theories of Concurrency: Unification and Extension, Amsterdam, The Netherlands, August 27-30, 1990, Proceedings*, volume 458 of *Lecture Notes in Computer Science*. Springer, 1990.

[BM01] J.C.M. Baeten and C.A. Middelburg. Process algebra with timing: Real time and discrete time. In *Handbook of Process Algebra*, pages 627–684. Elsevier, 2001.

[BM02] J.C.M. Baeten and C.A. Middelburg. *Process Algebra with Timing*. Springer, 2002.

[BM09] Ahmed Bouajjani and Oded Maler, editors. *Computer Aided Verification, 21st International Conference, CAV 2009, Grenoble, France, June 26 - July 2, 2009. Proceedings*, volume 5643 of *Lecture Notes in Computer Science*. Springer, 2009.

[BMU98] J.A. Bergstra, C.A. Middelburg, and Y.S. Usenko. Discrete time process algebra and the semantics of sdl. Technical report, CWI Report SEN-R9809, Centre for Mathematics and Computer Science, 1998.

[Bog07] Vladimir Bogachev. *Measure Theory*. Springer, 2007.

[BRRdS96] Amar Bouali, Annie Ressouche, Valérie Roy, and Robert de Simone. The fc2tools set. In Rajeev Alur and Thomas A. Henzinger, editors, *CAV*, volume 1102 of *Lecture Notes in Computer Science*, pages 441–445. Springer, 1996.

[Buc94] Peter Buchholz. Exact and ordinary lumpability in finite markov chains. pages 31–75:59–75, 1994.

[CE81] Edmund M. Clarke and E. Allen Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In Dexter Kozen, editor, *Logic of Programs*, volume 131 of *Lecture Notes in Computer Science*, pages 52–71. Springer, 1981.

[CGH+08] Nicolas Coste, Hubert Garavel, Holger Hermanns, Richard Hersemeule, Yvain Thonnart, and Meriem Zidouni. Quantitative evaluation in embedded system design: Validation of multiprocessor multithreaded architectures. In *DATE*, pages 88–89. IEEE, 2008.

[CHLS09] Nicolas Coste, Holger Hermanns, Etienne Lantreibecq, and Wendelin Serwe. Towards performance prediction of compositional models in industrial gals designs. In Bouajjani and Maler [BM09], pages 204–218.

[Cin75] Erhan Cinlar. *Introduction to Stochastic Processes*. Prentice-Hall Inc., 1975.

[CLM+04] Marcello Coppola, Riccardo Locatelli, Giuseppe Maruccia, Lorenzo Pieralisi, and Alberto Scandurra. Spidergon: a novel on-chip communication network. In *System-on-Chip, 2004. Proceedings. 2004 International Symposium on*, November 2004.

[Coh80] Donald L. Cohn. *Measure Theory*. Birkhäuser, 1980.

[CPS93] Rance Cleaveland, Joachim Parrow, and Bernhard Steffen. The concurrency workbench: A semantics-based tool for the verification of concurrent systems. *ACM Trans. Program. Lang. Syst.*, 15(1):36–72, 1993.

[dA97] Luca de Alfaro. *Formal Verification of Probabilistic Systems*. PhD thesis, 1997.

[DHK99] Pedro R. D'Argenio, Holger Hermanns, and Joost-Pieter Katoen. On generative parallel composition. *Electr. Notes Theor. Comput. Sci.*, 22, 1999.

[FPW05] Wan Fokkink, Jun Pang, and Anton Wijs. Is timed branching bisimilarity an equivalence indeed? In Paul Pettersson and Wang Yi, editors, *FORMATS*, volume 3829 of *Lecture Notes in Computer Science*, pages 258–272. Springer, 2005.

[FPW08] Wan Fokkink, Jun Pang, and Anton Wijs. Is timed branching bisimilarity a congruence indeed? *Fundam. Inform.*, 87(3-4):287–311, 2008.

[Gar89] Hubert Garavel. *Compilation et Vérification de Programmes LOTOS*. PhD thesis, 1989.

[GB87] Rob Gerth and Andy Boucher. A timed failures model for extended communicating processes. In Thomas Ottmann, editor, *ICALP*, volume 267 of *Lecture Notes in Computer Science*, pages 95–114. Springer, 1987.

[GcJS90] Alessandro Giacalone, Chi chang Jou, and Scott A. Smolka. Algebraic reasoning for probabilistic concurrent systems. In *Proc. IFIP TC2 Working Conference on Programming Concepts and Methods*, pages 443–458. North-Holland, 1990.

[GH02] Hubert Garavel and Holger Hermanns. On combining functional verification and performance evaluation using cadp. In Lars-Henrik Eriksson and Peter A. Lindsay, editors, *FME*, volume 2391 of *Lecture Notes in Computer Science*, pages 410–429. Springer, 2002.

[GHPS09] Hubert Garavel, Claude Helmstetter, Olivier Ponsini, and Wendelin Serwe. Verification of an industrial systemc/tlm model using lotos and cadp. In *7th ACM-IEEE International Conference on Formal Methods and Models for Codesign MEMOCODE'2009*, Cambridge, MA États-Unis d'Amérique, 2009.

[GL01] Hubert Garavel and Frédéric Lang. Svl: A scripting language for compositional verification. In Myungchul Kim, Byoungmoon Chin, Sungwon Kang, and Danhyung Lee, editors, *FORTE*, volume 197 of *IFIP Conference Proceedings*, pages 377–394. Kluwer, 2001.

[GLMS07] Hubert Garavel, Frédéric Lang, Radu Mateescu, and Wendelin Serwe. Cadp 2006: A toolbox for the construction and analysis of distributed processes. In Werner Damm and Holger Hermanns, editors, *CAV*, volume 4590 of *Lecture Notes in Computer Science*, pages 158–163. Springer, 2007.

[GS98] Hubert Garavel and Mihaela Sighireanu. Towards a second generation of formal description techniques - rationale for the design of e-lotos. In *FMICS*, pages 187–230, 1998.

[GV90] Jan Friso Groote and Frits W. Vaandrager. An efficient algorithm for branching bisimulation and stuttering equivalence. In Mike Paterson, editor, *ICALP*, volume 443 of *Lecture Notes in Computer Science*, pages 626–638. Springer, 1990.

[Han91] Hans A. Hansson. *Time and probability in formal design of distributed systems*. PhD thesis, docs, 1991. Available as report DoCS 91/27.

[Han94] Hans A. Hansson. *Time and Probability in Formal Design of Distributed Systems (Real-Time Safety Critical Systems, Vol 1)*. Elsevier Science & Technology, 1994.

[Hav00] Boudewijn R. Haverkort. Markovian models for performance and dependability evaluation. In Ed Brinksma, Holger Hermanns, and Joost-Pieter Katoen, editors, *European Educational Forum: School on Formal Methods and Performance Analysis*, volume 2090 of *Lecture Notes in Computer Science*, pages 38–83. Springer, 2000.

[Her02] Holger Hermanns. *Interactive Markov Chains - The Quest for Quantified Quality*. Springer, 2002.

[HHK+00] Holger Hermanns, Ulrich Herzog, Ulrich Klehmet, Vassilis Mertsiotakis, and Markus Siegle. Compositional performance modelling with the tipptool. *Perform. Eval.*, 39(1-4):5–35, 2000.

[HHK02] Holger Hermanns, Ulrich Herzog, and Joost-Pieter Katoen. Process algebra for performance evaluation. *Theor. Comput. Sci.*, 274(1-2):43–87, 2002.

[Hil96] Jane Hillston. *A compositional Approach to Performance Modelling*. Cambridge University Press, 1996.

[HJ90] Hans Hansson and Bengt Jonsson. A calculus for communicating systems with time and probabitilies. In *IEEE Real-Time Systems Symposium*, pages 278–287, 1990.

[HJ94] Hans Hansson and Bengt Jonsson. A logic for reasoning about time and reliability. *Formal Asp. Comput.*, 6(5):512–535, 1994.

[HJ03] Holger Hermanns and Christophe Joubert. A set of performance and dependability analysis components for cadp. In Hubert Garavel and John Hatcliff, editors, *TACAS*, volume 2619 of *Lecture Notes in Computer Science*, pages 425–430. Springer, 2003.

[HK00] Holger Hermanns and Joost-Pieter Katoen. Automated compositional markov chain generation for a plain-old telephone system. *Sci. Comput. Program.*, 36(1):97–127, 2000.

[HKNP06] Andrew Hinton, Marta Z. Kwiatkowska, Gethin Norman, and David Parker. Prism: A tool for automatic verification of probabilistic systems. In Holger Hermanns and Jens Palsberg, editors, *TACAS*, volume 3920 of *Lecture Notes in Computer Science*, pages 441–444. Springer, 2006.

[HL98] Holger Hermanns and Markus Lohrey. Priority and maximal progress are completely axiomatisable (extended abstract). In Davide Sangiorgi and Robert de Simone, editors, *CONCUR*, volume 1466 of *Lecture Notes in Computer Science*, pages 237–252. Springer, 1998.

[HNSY94] Thomas A. Henzinger, Xavier Nicollin, Joseph Sifakis, and Sergio Yovine. Symbolic model checking for real-time systems. *Inf. Comput.*, 111(2):193–244, 1994.

[Hoa78] C. A. R. Hoare. Communicating sequential processes. *Commun. ACM*, 21(8):666–677, 1978.

[HR95] Matthew Hennessy and Tim Regan. A process algebra for timed systems. *Inf. Comput.*, 117(2):221–239, 1995.

[HS99] Holger Hermanns and Markus Siegle. Bisimulation algorithms for stochastic process algebras and their bdd-based implementation. In Joost-Pieter Katoen, editor, *ARTS*, volume 1601 of *Lecture Notes in Computer Science*, pages 244–264. Springer, 1999.

[HT02] András Horváth and Miklós Telek. Phfit: A general phase-type fitting tool. In *TOOLS '02: Proceedings of the 12th International Conference on Computer Performance Evaluation, Modelling Techniques and Tools*, pages 82–91, London, UK, 2002. Springer-Verlag.

[INR] INRIA/VASY. Page de manuel de l'outil bcg_min. `http://vasy.inria.fr/cadp/man/bcg_min.html`.

[JAB01] S. A. Smolka Jan A. Bergstra, A. Ponse. *Handbook of Process Algebra*. Elsevier Science & Technology, 2001.

[JG89] M. Joseph and A. Goswami. Relating computation and time. Technical report, Coventry, UK, UK, 1989.

[JLY01] Bengt Jonsson, Kim G. Larsen, and Wang Yi. Probabilistic extensions of process algebras. In *Handbook of Process Algebra*, pages 685–710. Elsevier, 2001.

[JS90] Chi-Chang Jou and Scott A. Smolka. Equivalences, congruences, and complete axiomatizations for probabilistic processes. In Baeten and Klop [BK90], pages 367–383.

[KGN+09] Roope Kaivola, Rajnish Ghughal, Naren Narasimhan, Amber Telfer, Jesse Whittemore, Sudhindra Pandav, Anna Slobodová, Christopher Taylor, Vladimir Frolov, Erik Reeber, and Armaghan Naik. Replacing testing with formal verification in intel coretm i7 processor execution engine validation. In Bouajjani and Maler [BM09], pages 414–429.

[KNP04] Marta Z. Kwiatkowska, Gethin Norman, and David Parker. Prism 2.0: A tool for probabilistic model checking. In *QEST*, pages 322–323. IEEE Computer Society, 2004.

[KNP07] Marta Z. Kwiatkowska, Gethin Norman, and David Parker. Stochastic model checking. In Marco Bernardo and Jane Hillston, editors, *SFM*, volume 4486 of *Lecture Notes in Computer Science*, pages 220–270. Springer, 2007.

[KS76] John G. Kemeny and J. Laurie Snell. *Finite Markov Chains*. Springer, 1976.

[KSK76] John G. Kemeny, J. Laurie Snell, and Anthony W. Knapp. *Denumerable Markov Chains*. Springer, 1976.

[Kul95] Vidyadhar G. Kulkarni. *Modeling and Analysis of Stochastic Systems*. Chapman & Hall, 1995.

[Kur97] Robert P. Kurshan. Formal verification in a commercial setting. In *DAC*, pages 258–262, 1997.

[LHK01] Gabriel G. Infante López, Holger Hermanns, and Joost-Pieter Katoen. Beyond memoryless distributions: Model checking semi-markov chains. In Luca de Alfaro and Stephen Gilmore, editors, *PAPM-PROBMIV*, volume 2165 of *Lecture Notes in Computer Science*, pages 57–70. Springer, 2001.

[LL98] Luc Léonard and Guy Leduc. A formal definition of time in lotos. *Formal Asp. Comput.*, 10(3):248–266, 1998.

[LMdS08] Su-Young Lee, Frédéric Mallet, and Robert de Simone. Dealing with aadl end-to-end flow latency with uml marte. In *ICECCS*, pages 228–233. IEEE Computer Society, 2008.

[LS92] Kim Guldstrand Larsen and Arne Skou. Compositional verification of probabilistic processes. In Rance Cleaveland, editor, *CONCUR*, volume 630 of *Lecture Notes in Computer Science*, pages 456–471. Springer, 1992.

[Mil80] Robin Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer, 1980.

[Mil90] Robin Milner. Operational and algebraic semantics of concurrent processes. In *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B)*, pages 1201–1242. 1990.

[MT90] Faron Moller and Chris M. N. Tofts. A temporal calculus of communicating systems. In Baeten and Klop [BK90], pages 401–415.

[Neu81] M.F. Neuts. *Matrix-geometric Solutions in Stochastic Models-An Algorithmic Approach*. The Johns Hopkins University Press, 1981.

[Nor98] J. R. Norris. *Markov Chains*. Cambridge University Press, 1998.

[NS91] Xavier Nicollin and Joseph Sifakis. An overview and synthesis on timed process algebras. In Kim Guldstrand Larsen and Arne Skou, editors, *CAV*, volume 575 of *Lecture Notes in Computer Science*, pages 376–398. Springer, 1991.

[NS94] Xavier Nicollin and Joseph Sifakis. The algebra of timed processes, atp: Theory and application. *Inf. Comput.*, 114(1):131–178, 1994.

[OS06] Joël Ouaknine and Steve Schneider. Timed csp: A retrospective. *Electr. Notes Theor. Comput. Sci.*, 162:273–276, 2006.

[PA91] B. Plateau and K. Atif. Stochastic automata network of modeling parallel systems. *IEEE Transactions on Software Engineering*, 17(10):1093–1108, 1991.

[Pla85] Brigitte Plateau. On the stochastic structure of parallelism and synchronization models for distributed algorithms. In *SIGMETRICS*, pages 147–154, 1985.

[Pnu77] Amir Pnueli. The temporal logic of programs. In *FOCS*, pages 46–57. IEEE, 1977.

[Put94] Martin L. Puterman. *Markov Decision Processes*. Wiley, 1994.

[QS82] Jean-Pierre Queille and Joseph Sifakis. Specification and verification of concurrent systems in cesar. In Mariangiola Dezani-Ciancaglini and Ugo Montanari, editors, *Symposium on Programming*, volume 137 of *Lecture Notes in Computer Science*, pages 337–351. Springer, 1982.

[RdS90] Valérie Roy and Robert de Simone. Auto/autograph. In Edmund M. Clarke and Robert P. Kurshan, editors, *CAV*, volume 531 of *Lecture Notes in Computer Science*, pages 65–75. Springer, 1990.

[RR86] George M. Reed and A. W. Roscoe. A timed model for communicating sequential processes. In Laurent Kott, editor, *ICALP*, volume 226 of *Lecture Notes in Computer Science*, pages 314–323. Springer, 1986.

[RS59] M. O. Rabbin and D. Scott. Finite automata and their decision problems. *IBM Journal of Research and Development*, 3:114–125, 1959.

[Sch95] Steve Schneider. An operational semantics for timed csp. *Inf. Comput.*, 116(2):193–213, 1995.

[Seg95] Roberto Segala. *Modeling and Verification of Randomized Distributed Real-Time Systems*. PhD thesis, Cambridge, MA, USA, 1995.

[SL95] Roberto Segala and Nancy A. Lynch. Probabilistic simulations for probabilistic processes. *Nord. J. Comput.*, 2(2):250–273, 1995.

[Ste94] W.J. Stewart. *Introduction to the numerical solution of Markov chains*. Princeton University Press, 1994.

[Sub87] S. Purushothaman.and P.A. Subrahmanyam. Reasoning about probabilistic behavior in concurrent systems. *IEEE Trans. Softw. Eng.*, 13(6):740–745, 1987.

[TG08] Nikola Trcka and Sonja Georgievska. Branching bisimulation congruence for probabilistic systems. *Electr. Notes Theor. Comput. Sci.*, 220(3):129–143, 2008.

[Tij03] Henk C. Tijms. *A First Course in Stochastic Models*. Wiley, 2003.

[vdZ01] Mark van der Zwaag. The cones and foci proof technique for timed transition systems. *Inf. Process. Lett.*, 80(1):33–40, 2001.

[vGSS95] Rob J. van Glabbeek, Scott A. Smolka, and Bernhard Steffen. Reactive, generative and stratified models of probabilistic processes. *Inf. Comput.*, 121(1):59–80, 1995.

[vGW96] Rob J. van Glabbeek and W. P. Weijland. Branching time and abstraction in bisimulation semantics. *J. ACM*, 43(3):555–600, 1996.

[Yi91] Wang Yi. Ccs + time = an interleaving model for real time systems. In Javier Leach Albert, Burkhard Monien, and Mario Rodríguez-Artalejo, editors, *ICALP*, volume 510 of *Lecture Notes in Computer Science*, pages 217–228. Springer, 1991.

[Yov96] Sergio Yovine. Model checking timed automata. In Grzegorz Rozenberg and Frits W. Vaandrager, editors, *European Educational Forum: School on Embedded Systems*, volume 1494 of *Lecture Notes in Computer Science*, pages 114–152. Springer, 1996.

[ZN10] Lijun Zhang and Martin R. Neuhäußer. Model checking interactive markov chains. In Javier Esparza and Rupak Majumdar, editors, *TACAS*, volume 6015 of *Lecture Notes in Computer Science*, pages 53–68. Springer, 2010.

Notations

List of Abbreviations

MC	Markov Chain
DTMC	Discrete-Time Markov Chain
CTMC	Continuous-Time Markov Chain
AMC	Annotated Markov Chain
DTAMC	Discrete-Time AMC
CTAMC	Continuous-Time AMC
SMC	Semi-Markov Chain
MDP	Markov Decision Process
IMC	Interactive Markov Chain
IPC	Interactive Probabilistic Chain

Graphs

$\mathcal{M}$	a Markov chain
$\mathcal{M}_P$	a probabilistic chain
$\mathcal{M}_M$	a Markovian chain
$\mathscr{A}$	an annotation function
$\mathcal{M}_P^{\mathscr{A}}$	an annotated probabilistic chain
$\mathscr{D}$	the set of all DTAMCs
$\mathcal{M}_S$	an SMC
P	an IPC
M	an IMC
$\mathscr{P}$	the set of all IPCs
$\mathscr{M}$	the set of all IMCs
$P_{\not\to_\tau}$	a maximal progress-cut IPC
$P_{\not\to_A}$	an urgency-cut IPC
$M_{\not\to_\tau}$	a maximal progress-cut IMC
$M_{\not\to_A}$	an urgency-cut IMC
s	a state
S	a set of states

Transitions

$\mathcal{A}$	a set of action including the silent action τ
e	a regular expression over $\mathcal{A}$
$\mathfrak{E}$	the set of regular expressions over $\mathcal{A}$
$\mathfrak{e}$	the null regular expression
$\mathfrak{S}(e)$	the set of actions composing the regular expression e
$\xrightarrow{a}$	an interactive transition with $a \in \mathcal{A}$
$\xrightarrow{e}$	a sequence of interactive transitions identified by the regular expression e
$\overset{p}{\rightsquigarrow}$	a probabilistic transition ($p \in]0,1]$)
$\overset{\lambda}{\dashrightarrow}$	a Markovian transition ($\lambda \in]0,+\infty]$)
$\overset{a\ p}{\square\!\!\rightsquigarrow}$	a labeled probabilistic transition ($a \in \mathcal{A}$ and $p \in]0,1]$)
$\xrightarrow{\tau*}$	a transitive closure of τ-transitions
$\mathcal{D}$	a time probabilistic distribution over a labeled probabilistic transition
$\mathbf{C}_n$	a constant distribution (equal to n) over a labeled probabilistic transition

Sets and Multisets

$\mathscr{S}$	an arbitrary set
$\mathscr{S}_\times$	an arbitrary multiset
$\mathcal{P}(\mathscr{S})$	the power set of $\mathscr{S}$
$\mathcal{P}^\times(\mathscr{S})$	the power multiset of $\mathscr{S}$
$\sum \{\!] \mid [\!\}$	the sum over a multiset
$\mathsf{card}(\mathscr{S})$	the cardinal of the set $\mathscr{S}$
$\mathsf{gcd}\left\{\mathscr{S}\right\}$	The greater common divider of numbers in the set $\mathscr{S}$

Equivalence Relations

$\mathcal{R}$	an arbitrary relation
$\mathscr{E}$	an equivalence relation
$\sim_P$	a strong bisimulation over $\mathscr{D}$
$\sim_M$	a strong bisimulation over the set of all CTAMCs
$\sim$	a strong bisimulation over $\mathscr{M}$ or $\mathscr{P}$
$\approx$	a branching bisimulation over $\mathscr{M}$ or $\mathscr{P}$
$\mathcal{C}$	an equivalence class
$\mathscr{P}/_{\mathscr{E}}$	the set of equivalence classes induced by $\mathscr{E}$ on $\mathscr{P}$
$[s]_{/_{\mathscr{E}}}$	the equivalence class of s w.r.t. $\mathscr{E}$
$P_{/_\approx}$	the quotient of P w.r.t. $\approx$

Measure Theory

$\mathcal{F}$	a field
σ	an infinite path in a DTAMC or an IPC
$\vec{\sigma}$	a finite path in a DTAMC or an IPC
$Paths(s)$	the set of all paths starting in state s
$Paths^{\vec{}}(s)$	the set of all finite path starting in s
$last(\vec{\sigma})$	the last state of the finite path $\vec{\sigma}$
$\lvert\vec{\sigma}\rvert$	the length of the finite path $\vec{\sigma}$
$\circ$	concatenation operator for paths
$\mathrm{tr}(\vec{\sigma})$	the trace of the path σ
$\mathrm{tr}_\tau(\vec{\sigma})$	the abstract trace of the path σ
$\mathscr{s}$	a scheduler
$\xi_{\mathscr{s}}$	a path expander w.r.t. $\mathscr{s}$
$\Xi_{\mathscr{s}}$	a silent path expander w.r.t. $\mathscr{s}$