

Automation of Importance Splitting Techniques for Rare Event Simulation

Carlos E. Budde



Automation of Importance Splitting Techniques for Rare Event Simulation

by

Carlos E. Budde



Automation of Importance Splitting Techniques for Rare Event Simulation,
by Carlos E. Budde, is distributed under the Creative Commons License
Attribution-NonCommercial 2.5 Argentina.

Automation of Importance Splitting Techniques for Rare Event Simulation

by

Carlos E. Budde

May 2017

Advisor: Pedro R. D'Argenio

Co-advisor: Holger Hermanns

Presented to the Facultad de Matemática, Astronomía, Física y
Computación as part of the requisites for obtaining the degree of
Doctor in Computer Sciences of the

Universidad Nacional de Córdoba



CCS Concepts: • **Computing methodologies** → **Rare-event simulation**; *Discrete-event simulation*; Modeling and simulation; Model verification and validation.

Other keywords: formal verification of systems, model analysis via simulation, rare event simulation, importance splitting, RESTART, automatic importance splitting.

Abstract

Many efficient analytic and numeric approaches exist to study and verify formal descriptions of probabilistic systems. *Probabilistic model checking* is a prominent example, which can handle several modelling formalisms through various study angles and degrees of detail. However its core resolution algorithms depend on the memoryless property, meaning only Markovian models can be studied, with few limited exceptions. Furthermore the state-space of the model needs to fit in the physical memory of the computer.

Discrete-event Monte Carlo simulation provides an alternative for the generality of automata-based stochastic processes. The term *statistical model checking* has been coined to signify the application of simulation in a model checking environment, where systems are formally described and properties written in some temporal logic (LTL, CSL, PCTL*, etc.) are answered within the confidence criteria requested by the user.

Such simulation approaches can however fail yielding no answer to the query. This typically happens when statistic analysis of the paths generated shows the data available is insufficient to meet the requested confidence criteria, and then more simulation is needed. When the value to estimate depends on the occurrence of rare events, viz. events which are seldom observed in the normal operation of the system, the situation degenerates to infeasible requirements, e.g. two months of standard Monte Carlo simulation may be needed to provide the desired 90% confidence interval.

Specialised simulation strategies exist to combat this problem, which lower the variance of the estimator and hence reduce simulation time. Importance splitting is one such technique, which requires a guiding function to steer the generation of paths towards the rare event. This *importance function* is typically given in an ad hoc fashion by an expert in the field of the model under study. An inadequate choice may lead to inefficient simulation and long computation times.

This thesis presents automatic approaches to derive the importance function, based on a formal description of the model and of the property to estimate. Since the basis of estimations is discrete event simulation, general stochastic processes can be covered with these approaches. The modelling formalism is Input/Output Stochastic Automata (IOSA, [DLM16]) and both transient and steady-state (probabilistic) properties involving rare events

can be estimated. Since IOSA is a modular formalism, the efficiency of two different techniques has been studied: deriving the importance function from the fully composed model, and deriving it locally in the individual system modules. The latter option alleviates some memory issues but requires composing the locally generated functions into a global importance function, which provided another subject of research also included in this thesis.

Prototypical yet extensible tools have been implemented to test the feasibility and efficiency of these automatic techniques which face the rare event simulation problem. Some insight into their implementation and the results of experimentation are presented in the thesis.

Acknowledgments

So much to say, anything will run short; but these people deserve the effort. I believe anyone who has had the tenacity—and good luck—to culminate his Ph.D. studies, must recognise the human sustainment that made it possible.

Support came from everywhere but mostly from my family, who were there in the good days and in the bad days. My mother Lucía and my brother Leopoldo are foundations without which this structure would have never been. My late father Carlos is here as well, and always will be. As a younger man I thought that since family (whatever that means for each person) is so close to us, we cannot fully appreciate the extent to which we rely on it. I am older now, so allow me to repeat myself: without you, this would not be.

There is of course a bigger family: my aunt Luisa, Pablo, the Di Fiori, Uacha and Etruria. There is, furthermore, family we find along the way: Lichi, Zerep, la Mari, CN1, FAMAF, Muay-Thai, my krus Sergio and Pablo, you Pao . . . I am happily certain that these bonds can only grow stronger.

I am not forgetting about you, Pedro, who helped me so much and gave me some hard times too. Neither will you be left out, Raúl, with whom I shared questions, code, and mate, and who will soon be writing his own thesis. Looking back on all we built together, pucha, it ain't so little after all.

There are also lots of people in FAMAF who helped me in my studies: Nico, Pedro S.T., Charly, Oscar, Damián, Laura, Silvia, Pablo, Félix, . . . the list is endless. Plus the Saarbrücken team: my co-advisor Holger, the great Arnd Hartmanns, whose intellect and friendship are a lighthouse in the stormy seas of research; there are also Gilles, Luis, Yuliya, Hassan, and many more. Our paths will keep merging, I am confident of that.

I also want to thank José Villén-Altamirano, who gave me an invaluable hand in understanding the subtler mechanisms which consolidate RESTART, showing me at the same time how hospitable the Spanish can be.

There are people implicitly mentioned here, whose names do not appear. Yet the contents of this thesis must begin, so let me add without further ado:

Thank you all! This work is dedicated to you.

Agradecimientos

Con tanto por decir todo intento quedará trunco; pero esta gente lo vale. Creo que cualquier persona que haya tenido la tenacidad—y buena fortuna—de culminar su doctorado, debe reconocer el factor humano que lo hizo posible.

El apoyo que recibí provino de muchas fuentes, pero principalmente de mi familia. Mi madre Lucía y mi hermano Leopoldo son cimientos sin los cuales jamás habría podido erigir esta estructura. Mi padre Carlos también está conmigo, y siempre lo estará. Antes pensaba que la familia (cualquiera sea el significado que cada persona le otorgue a esta palabra) nos es tan cercana, que no podemos apreciar del todo lo indispensable de su presencia. Ya soy más viejo, por lo que repito: sin ustedes esto no existiría.

Hay también una familia más grande: mi tía Luisa, el Pablo, los Di Fiori, Ucacha y Etruria. Hay, a su vez, familia que encontramos en el camino: Lichi, Zerep, la Mari, CN1, FAMAF, Muay-Thai, mis krus Sergio y Pablo, la Pao ... Tengo la feliz certeza de que estos vínculos sólo pueden afianzarse.

No me olvido de vos, Pedro, que tanto me ayudaste y tanto me hiciste renegar también. Ni de vos, Raúl, con quien compartí dudas, código, y mate, y quien pronto estará escribiendo su propia tesis. Mirando en retrospectiva lo que construimos juntos, pucha, no es tan poco al fin de cuentas.

Hay mucha gente en FAMAF que me dio una mano invaluable: Nico, Pedro S.T., Charly, Oscar, Damián, Laura, Silvia, Pablo, Félix, ... la lista es larga. Saarbrücken también contribuyó: están mi co-director Holger, y el gran Arnd Hartmanns, cuyo intelecto y amistad son un faro en los tormentosos mares de la investigación; están también Gilles, Luis, Yuliya, Hassan, y muchos más. Nuestros caminos se seguirán cruzando, de eso estoy seguro.

Quiero agradecer especialmente a José Villén-Altamirano, quien me ayudó muchísimo a entender los mecanismos más sutiles que consolidan a RESTART, mostrándome al mismo tiempo cuán hospitalarios pueden ser los Españoles.

Hay personas con mención implícita cuyos nombres no aparecen aquí. Sin embargo los contenidos de la tesis deben comenzar, por lo que añado sin más:

¡Gracias a todos! Les dedico esta tesis a ustedes.

Contents

1	Introduction	1
1.1	Motivations and goals	4
1.2	Related work	6
1.3	Contributions and outline of the thesis	8
2	Background	12
2.1	System modelling	12
2.2	Model property queries	19
2.3	Analysis of the model	23
2.3.1	Overview and known approaches	24
2.3.2	Simulation	27
2.3.3	Estimation	29
2.3.4	Convergence and stopping criteria	31
2.4	Rare events	35
2.5	Importance splitting	39
2.5.1	General splitting theory	39
2.5.2	Variants of the basic technique	47
2.6	RESTART	52
2.7	Applicability and performance of I-SPLIT	57
3	Monolithic I-SPLIT	61
3.1	The importance of the I-FUN	61
3.2	Deriving an importance function	68
3.2.1	Objective	68
3.2.2	Formal setting	69
3.2.3	Derivation algorithm	70
3.3	Implementing automatic I-SPLIT	76
3.3.1	Modelling language	76
3.3.2	User query specification	84
3.3.3	Selection of the thresholds	85
3.3.4	Estimation and convergence	90
3.4	Tool support	93
3.5	Case studies	98
3.5.1	Experimentation setting	98

3.5.2	Tandem queue	99
3.5.3	Discrete time tandem queue	103
3.5.4	Mixed open/closed queue	106
3.5.5	Queue with breakdowns	110
3.6	Limitations of the monolithic approach	114
4	Compositional I-SPLIT	119
4.1	The road to modularity	119
4.2	Local importance function	121
4.2.1	Projection of the rare event	121
4.2.2	Algorithms and technical issues	124
4.3	Global I-FUN composition	128
4.3.1	Basic strategies	129
4.3.2	Monolithism vs. compositionality	132
4.3.3	Rings and semirings	136
4.3.4	Post-processing the functions	139
4.4	Input/Output Stochastic Automata	140
4.5	Automation and tool support	146
4.5.1	Selection of the thresholds	146
4.5.2	IOSA model syntax	150
4.5.3	The FIG tool	153
4.6	Case studies	160
4.6.1	Experimentation setting	160
4.6.2	Tandem queue	162
4.6.3	Triple tandem queue	167
4.6.4	Queue with breakdowns	170
4.6.5	Database system	172
4.6.6	Oil pipeline	176
5	Final remarks	189
5.1	Future work	190
Appendix A	System models	193
A.1	Tandem queue (PRISM)	193
A.2	Discrete time tandem queue (PRISM)	194
A.3	Mixed open/closed queue (PRISM)	196
A.4	Queue with breakdowns (PRISM)	196
A.5	Database system (PRISM)	198
A.6	Tandem queue (IOSA)	200

A.7 Tandem queue' (PRISM)	201
A.8 Triple tandem queue (IOSA)	202
A.9 Queue with breakdowns (IOSA)	204
A.10 Database system (IOSA)	206
A.11 Oil pipeline (IOSA)	209
Appendix B Measure theory	212
Appendix C Nondeterministic Labelled Markov Processes	215
Bibliography	218

Introduction

1

It is deeply rooted in human nature, providing such a thing exists, to study and modify our environment in an attempt to minimise threats and increase our chances of survival and comfort. In an ever increasingly technological and electronic society, these attempts materialise in the development of information storage and computation systems. These computer-based processes and tools can become extremely complex, and since our well-being depends on them, they are under continuous human and automated revision to ensure their proper functioning.

Examples of such undertakings are ubiquitous: from regular *mechanical checks in trains*, or verifications performed in a newly written *piece of code*, all the way up to the highly structured protocols involved in every assembly phase of a *spacecraft*.

In spite of these efforts, the inextricable foundations of reality make it impossible to completely avoid accidents. Either by human error or machinery malfunction, the 22nd of February 2012 “*la tragedia de Once*” (the Once—a train station’s nickname—Tragedy) took the life of more than fifty people, in the worst Argentinian train mishap of the last thirty years.

Undesired outcomes are also observed in processes isolated from a hostile natural environment. Consider *Heartbleed*, the security bug in the OpenSSL cryptography library, used worldwide to secure the most important value of western civilization: private capital. The source code was a peer-reviewed implementation of a standardised protocol, yet it contained a flaw which could infringe the user’s privacy by allowing a buffer over-read. This vulnerability was subject to massive broadcast, and the Codenomicon company provided the bug with a logo of its own—see Figure 1.1.

Even in highly protocolarised production chains do these bugs find a crack to hide in, coming out to cause mayhem in obnoxious ways. Space shuttle programs are famous for the thoroughness of their security checks and controlled procedures. Be that as it may, the *Space Shuttle Columbia disaster* destroyed seven lives and millions of dollars of investment and research, in

an accident that slipped the mind of technicians and engineers at NASA.



Figure 1.1: Heartbleed logo[†]

There is no denying the limits of human revision. Inspections can be carried out, protocols followed, code reviewed; but the subjective factor, that unmeasurable injector of failures, will be present as long as humans are involved in the process. That is why formal guarantees have gained in popularity for the last quarter of a century [CW96]. From the rigorousness of logic and mathematics, developing techniques to ensure that a *model of our system* satisfies certain vital properties, is not only beneficial but also increasingly necessary in the modern world. Two from the three incidents mentioned took place less than six years ago, accounting for the currency of the claim.

Model checking is a prominent example of one such technique; it is a verification procedure based on an exhaustive exploration of the state space of a model of the system [CES86, BK08, Har15]. The user provides such model and a formalisation of the property to be verified, and model checking replies whether the property holds (typically qualitative queries), and in certain scenarios it can measure to which extent does it hold (quantitative queries).

Nevertheless, the results of such formal proofs are only valid for the models where they were proved or verified. Thus the more realistic the model, the more useful the result. This has led from the initial discrete and deterministic settings of process algebra and transition systems, to the inclusion of nondeterministic behaviour, discrete probabilities, continuous time, and even (continuous) stochastic behaviour.

The price to pay for such complexities are more involved verification techniques and an ever increasing state space, whose storage in physical computer memory easily becomes infeasible. Focusing primarily on the dimension of the state space, several reduction procedures are currently known to work on a smaller abstraction of the original model description. Examples of such techniques include program slicing [Wei84], partial order reduction [Val90], confluence reduction [BvdP02], and several refinements of these as well as other strategies [Bry86, CFM⁺93, dAKN⁺00, DJJL02, DN04, BGC04]. Many involve performing verifications on a reduced model related to the original

[†] By Leena Snidate / Codenomicon - <http://heartbleed.com/heartbleed.svg>

one by means of a *bisimulation relation* [Mil89]. Unfortunately and more often than not, the theoretic hypotheses on which such techniques are founded can be quite restrictive, e.g. describing stochastic behaviour solely with memoryless probability density functions [BK08]. Another known issue that several minimisation procedures suffer from is requiring access to the full reachable state space [Har15], which results in alleviated verification times but certainly does not solve the state dimension problem.

There is a different approach, popularly known as *statistical model checking* [YS02,LDB10], which can operate without such problematic exploration of the full state space. This technique, quite distinct from the previously mentioned *standard* model checking, is based on the randomised production of system (model) executions. Each execution produced is interpreted as a new, independent sample of the behaviour of the system, stored to augment a *random sample*. This random sample is then statistically analysed to provide the user with a tentative answer to the query.

The nature of this answer is thus very different from the one produced by standard model checking, which is certain (or at the very least it is certain that it is not certain) of its final statement [BK08]. Instead, statistical model checking yields an estimate of the answer, which the user can rely on with certain measurable notion of confidence. Owing to its statistical origins, this estimate is usually provided in the form of an interval, within which the true value of the user’s property query is supposed to lie. Thus the user can request tighter intervals to be produced with higher confidence, whenever a more reliable answer to the query is desired [LDB10].

Since samples are produced and analysed on the fly, this approach does not need to represent the full state space of the system model. Hence, the issues related to having huge state spaces are avoided. Of course this does not come for free, and is paid back with usually longer computation times, related to the production of the system execution paths. It can happen that a huge number of fresh samples provides little new information, and thus estimations progress at a slow pace. This situation is exacerbated when the property under study depends on the observation of a *rare event*, whose occurrence is very unlikely in randomly produced paths. There is a whole research field, known as *rare event simulation*[‡], whose specific aim is to counter such detrimental scenarios [RT09b]. This will be the target field of the thesis.

[‡] Notice “simulation” here stands for the randomised generation of system execution paths; it is not related to the previously mentioned notion of “bisimulation.”

1.1 Motivations and goals

Two techniques stand out in the field of rare event simulation: importance sampling and importance splitting. *Importance sampling* [GI89, Hei95, JS06] fiddles with the stochastic behaviour of the system *tractably*, meaning that the modifications applied to the original probability distributions can be countered once an estimate is obtained, correcting any bias introduced. This way the chances of observing the rare event in randomly generated paths are increased, and estimations progress at a more reasonable pace. *Importance splitting* [KH51, Bay70, VAVA91] leaves the original model untouched and pursues a different goal, cloning promising simulations (e.g. execution paths) which are likely to produce a rare event, and truncating those which go astray. Therefore most of the computing effort is spent producing samples rich on information.

Each technique has its advantages and its drawbacks, and they complement each other in certain ways, as it is further discussed in Section 2.4. However, the tractability of the change of measure required by importance sampling [LMT09] is hard to perform in a systematic way. Even casting automation aside and conforming ourselves with ad hoc approaches, most known efforts are bent to the study of Markovian systems, due to the hardships of coming up with an efficient and tractable change of measure. See e.g. [GSH⁺92, LT11] and [LMT09]. This is at odds with the general motivations of the thesis, which we describe next, ergo we will focus on importance splitting.

In most standard model checking procedures, the so called *push-button approach* is one of the major appeals: once the model has been built and the property queries specified, the user can obtain the desired answers in a fully automatic way. We consider this a clear advantage of the method over other formal techniques such as theorem proving. Therefore, we wish to develop procedures which go as close as possible to such full automation.

However, standard model checking suffers from the infamous *state explosion problem*, which forces implementers to apply reduction by bisimulation and other such strategies. It is paramount to shrink the representation of the model, forcing it to fit in the physical memory of a computer, in order to apply the verification algorithms. We prefer to avoid this problem altogether, resorting to model analysis by simulation (i.e. statistical model checking).

There is another advantage in choosing simulation over standard model checking, related to the scope of model types covered by each approach. As a rule, we want to be as general as possible. Earlier model checking algorithms could only cope with Markovian systems, which is far too restrictive for our

intentions. The situation has changed over the years, deriving in a multiple-formalism, multiple-solution situation—see “The MODEST Toolset” in [Har15]. Yet in contrast, *if one leaves nondeterminism aside*, the simulation approach can be trivially extended to cope with any type of probabilistic, timed, or (continuous) stochastic behaviour. That counts as a further motivation: our final product should be easy to apply to as many models as possible.

Moreover, we are interested in the challenges posed by system analysis under a rare event regime. This means path generation cannot be carried out in the standard Monte Carlo fashion, lest the estimation procedures take too long to converge to a reasonable result. In that respect we concern ourselves with efficient simulation techniques, more specifically with importance splitting, because we believe it matches our interests best.

Summing up, the general motivations of the thesis involve the development of automatic techniques for system (model) analysis, using simulation and statistical analysis of execution paths. The systems modelled should be as general as possible, but the properties studied must involve some rare event, seldom observed when generating the paths. Also and more specifically, we wish to focus our studies on perfecting the importance splitting technique, harmonising it with these motivations.

The efficiency gain derived from the use of importance splitting lies in a proper selection of the *importance function* [VAVA91, VAVA02, Gar00, LLGLT09]. This function decides which simulation paths are striving near the rare event and which are deviating from it. Thus, overlooking some technical details, we can think that choosing an efficient importance function is equivalent to having a good implementation of importance splitting.

When approaching rare event simulation with importance splitting, it is customary to have the user provide an ad hoc importance function, together with the system model and property queries [VAVA91, CAB05, LLGLT09]. However and in view of the general motivations above, we would like to automate the construction of such function, with no user intervention in the process.

Besides it is noteworthy that several studies from the rare event literature, most prominently those concerning importance sampling, formalise a measure of the efficiency of their approach. Such studies are keen on developing optimal or asymptotically efficient (also known as logarithmic efficiency [LMT09]) implementations of their methods [GSH⁺92, GHSZ98, KN99]. Generally speaking it is helpful to count with rare event simulation mechanisms exhibiting

such properties, since they are guaranteed to converge fast—or as fast as possible—regardless of how rare this elusive event becomes.

Unfortunately, such studies adequate their endeavours to the specific systems under study, coming out with strong hypotheses which rule out generalisations. This is unavoidable when one desires to obtain such efficiency. Optimality requires a formal proof that the variance of the estimator is minimal in the given setting. Asymptotic (or logarithmic) efficiency requires formal proof that such variance grows polynomially as the rarity of the event grows exponentially—see Section 2.4. Hence these results must be moulded to fit the specific system under study, which goes against the general motivations mentioned above.

In view of the last remarks we list the specific goals sought in this thesis:

- developing algorithms to build the importance function used by the importance splitting technique,
 - ▷ these algorithms should take as input the same data provided to perform standard model analysis by simulation;
- embedding this function in a procedure, automated to the push-button extent, which implements importance splitting;
- building a software tool which implements this automatic procedure;
- giving empirical proof of the efficiency of our approach,
 - ▷ our implementation intends to be more efficient than the standard Monte Carlo approach,
 - ▷ neither optimality nor asymptotic efficiency are sought,
 - ▷ when feasible, results should be validated against verified data,
 - ▷ experimentation should be carried out in diverse models, including non-Markovian systems.

1.2 Related work

We are aware of a number of studies in roughly the same direction than ours. First and foremost [JLS13] share several of our general motivations. They also propose to derive the importance function, called *score function* in [JLS13, JLST15], from the same user input that statistical model checking

requires. They focus on the property query, which needs to be restated in an equivalent “layered” way. Thus the importance value (score) of a system state is related to the number of layers of the property that it satisfies.

This idea pays little or no attention to the specific system under study when deriving the score function. We believe that the structure of the model should also be taken into consideration when deriving such function. Furthermore, if the property query does not support the layered restatement [JLS13] propose, approximate heuristics must be used.

In [ZM12] and [RdBSH13] the modelling formalism is *Stochastic Petri Nets* (SPN). Both works use the structure of the net to boost simulations in a rare event regime; a comparison between them can be found in [ZRWCL16]. In particular, [ZM12] derives a heuristic to measure (roughly) the distance of an arbitrary marking from the markings that satisfy the property query. That is used to derive an importance function, in an approach resembling the one from Chapter 3 in this thesis.

However, certain decisions made by [ZM12] are reached through the use of Linear Programming, applicable to a restricted class of SPN (the *freely related T-semiflows* class, according to [ZRWCL16]). These decisions involve key aspects like the selection of the *splitting factor* and the *thresholds* for the application of RESTART (a particular importance splitting mechanism). As mentioned in the general motivations, this thesis aims at a broader scope of applicability. Otherwise, letting aside the use of SPN, the approach from [ZM12, Sec. IV] has certain similarities with our proposal in Section 3.2.3.

[RdBSH13] study importance sampling rather than importance splitting, although they claim that the distance function derived with their method could also be used for importance splitting. They apply the approach from Booth & Hendriks as reported in [LDT07], measuring the distance between a marking and the rare event. This way they achieve a speedup in the simulation of rare events without generating the entire state space.

The approach developed in [RdBSH13] is certainly elegant, but it relies on: dealing with Markovian firing delays exclusively; parameterizing all transitions intensities by some rarity parameter; and solving several Integer Linear Programming instances (known to be an NP-complete problem). They do not report simulation times in that work, even though they do in [ZRWCL16], showing an effective application of their strategy. Still, in that same work they report computation problems for larger model sizes. Besides they are restricted to Markovian SPN, and a specific goal of this thesis is to consider non-Markovian systems.

[Bar14] focuses on SPN and importance sampling as well. In other respects,

many of his motivations and goals coincide with the ones from this thesis. Restricting his studies to the Markovian world, Barbot's Ph.D. thesis gives formal proof of variance reduction in several distinct settings. Furthermore, in the last part of [Bar14], Barbot exemplifies the efficiency of his proposal empirically, running an importance sampling benchmark with a software tool that implements his technique.

Last, we notice that these works (just like this thesis) are based on a static analysis of the model and/or property query provided by the user. Instead [GVOK02] assign importance to the states (i.e. build the importance function) applying reversed simulation sequentially on all the states of the system. This requires some knowledge on the stationary distribution of the model, and the applicability of the approach is shown for finite discrete-time Markov chains.

1.3 Contributions and outline of the thesis

Besides this introduction, the conclusions, and some final appendices, this thesis is organised in three extensive chapters.

Chapter 2 covers the fundamental theoretic aspects required to follow the thesis. The chapter is mostly self-contained, aside from some references to the appendices. More precisely:

Sections 2.1 and 2.2 give an overview of several modelling formalisms and temporal logics, used to query the properties exhibited by a system model.

Section 2.3 studies some known techniques to automate the verifications and checks on such system models. Using the general motivations from Section 1.1 as our north, we pick our way through a variety of strategies and algorithms. In doing so we identify the strengths and weaknesses of each technique w.r.t. our application intentions.

Section 2.4 gives a formal introduction to the field of rare event simulation, and motivates the choice of a stopping criterion for estimations, later used during experimentation. This section justifies the transition from the general scope of sections 2.1 to 2.3, to the more specific field of sections 2.5 to 2.7.

Sections 2.5 and 2.6 first introduce importance splitting formally, then show a broad overview of available implementations, and finally focus on the technique we will use later during experimentation.

Section 2.7 probes the boundaries of importance splitting and identifies some open problems in the field. Its final discussion links all the notions presented along the chapter with some specific goals of the thesis.

Chapter 3 presents our first (monolithic) approach, from the original ideas that motivated it, to the numerical results of the experimentation on case studies taken from the literature. This is developed as follows:

Section 3.1 reflects on how critical the role of the importance function is, in order to obtain a good implementation of importance splitting. Examples are used to introduce the sensitive topics, which are then taken into account in the following sections.

Section 3.2 presents our first algorithm, devised to fulfill the first specific goal detailed in Section 1.1: deriving an importance function from other user input. The application setting is stated formally and a proof of termination for the algorithm is provided.

Section 3.3 develops a framework to implement an automatable importance splitting application. This is carried out from a monolithic-model stand, inherent to the algorithm presented in Section 3.2. Particularly Section 3.3.3 introduces the algorithm we use to select the thresholds required by the splitting simulations. All this fulfills our second specific goal.

Sections 3.4 and 3.5 move to the empirical realm, introducing the first software tool implemented during the development of this thesis. The tool is used in Section 3.5 to experiment on several Markovian case studies taken from the literature. The results of these experiments served to validate the correct functioning of the tool, and to give practical demonstration of the efficiency of our approach. Thus the two last specific goals are fulfilled, though the second one only partially (it would remain to experiment on non-Markovian systems).

Section 3.6 concludes analysing certain limitations of the approach proposed in this chapter. The most serious one is the need to generate the entire state space of a fully composed model, inherent to the monolithic nature of the approach. Though most of our goals are satisfactorily met by Chapter 3, the issue mentioned is quite restrictive. This compelled us to strive for solutions, which evolved into the research presented in Chapter 4.

Chapter 4 introduces a second (compositional) approach to automate importance splitting, attempting to solve or at least mitigate the issues

incurred by the monolithic strategy used in Chapter 4. The main topics covered in this chapter are organised as follows:

Section 4.1 explores the foundations of a compositional approach. It discusses certain aspects to be covered when deriving an importance function of distributed nature, stating two concrete challenges.

Sections 4.2 and 4.3 answer the challenges from Section 4.1, achieving the first specific goal of Section 1.1 in this new setting, viz. deriving a compositional importance function. More specifically, Section 4.2 shows how to decompose the (global) property query in order to build importance functions local to each system component. An algorithm is provided, which fits in the general framework from Chapter 3. Then Section 4.3 presents several strategies to re-compose the resulting set of local importance functions, in order to obtain a global function to be used during simulations. Section 4.3 also features a comparison between the monolithic approach from the previous chapter, and the compositional approach from this one.

Section 4.4 presents a newly developed modelling formalism, named IOSA, which drops completely the Markovian restrictions from the one employed in Chapter 3. This formalism is the basis upon which all the practical applications of Chapter 4 are built.

Section 4.5 casts the proposals and results from the previous sections into an empirical setting. Namely, the IOSA formalism from Section 4.4 is given a concrete syntax in Section 4.5.2, and Section 4.5.3 presents the second software tool developed in this thesis. The second and third goals from Section 1.1 are thus tackled in this section.

Section 4.6 provides empirical proof of the applicability and efficiency of the compositional approach developed in this chapter. Since IOSA tolerates arbitrary stochastic distributions, some of the models studied are not Markovian, achieving thus our final specific goal to its full extent.

Chapter 5 gives some final remarks on the general outcomes of this work, and mentions possible continuations to improve on them and to extend the applicability of our proposals.

Appendices A to C are addendums which contribute to the reproducibility of our experiments, and which briefly review the more formal notions behind some theories on which this thesis relies. Namely:

Appendix A includes the code of all the system models used to produce the numeric results presented. Two modelling languages are used: models from Chapter 3 are expressed in the PRISM input language; models from Chapter 4 are expressed in the IOSA model syntax.

Appendix B includes some elemental definitions and results from measure theory, which are required to comprehend Appendix C and the more formal aspects of Chapter 2.

Appendix C presents the basic notions of the NLMP formalism, which is used as semantic basis for the IOSA formalism employed in Chapter 4.

Background

2

This chapter briefly covers the fundamentals required to follow the thesis. Readers interested in a deeper understanding of the subjects here introduced can find some excellent reading material in:

- *Principles of Model Checking*, by Christel Baier and Joost-Pieter Katoen, [BK08], where several aspects of system modelling and verification are explained on rock-solid mathematical and computational grounds;
- *Rare Event Simulation using Monte Carlo Methods*, edited by Gerardo Rubino and Bruno Tuffin, [RT09b], a monograph on rare event simulation (RES) result of a collaborative effort by chief contributors to the field;
- *The splitting method in rare event simulation*, Marnix Garvels' Ph.D. thesis, [Gar00], featuring an in-depth analysis on the application of importance splitting techniques to solve the RES problem.

It is recommended to at least skim through this chapter, even when the reader feels a strong confidence in the subjects it introduces. The intention is not only to present the necessary theoretical background, but also to review the concepts and open problems that motivated the thesis. This is exposed in a way that gradually converges from the generality of formal modelling and verification, to the derivation of importance functions for applying multilevel splitting to the rare event simulation problem.

2.1 System modelling

There is an approach to study and understand the systems we devise, which has the appealing benefit of (partial) automatization and which provides guarantees of the results it yields: *formal modelling and verification*. To engage in this approach the core functionality of the system needs to be interpreted and described in terms of some formal language, which comprises

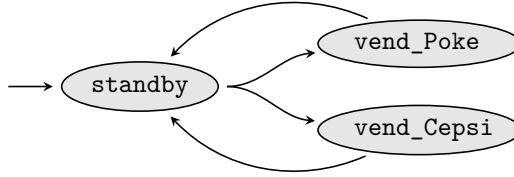


Figure 2.1: Soda vending machine (pirate version)

the non-automatable phase. Such abstraction task is by no means trivial, one of whose many difficulties lies in the choice of the relevant components and behaviour which are to be included in the abstraction. However the rewards compensate the effort: once the formal model is finished many studies can be carried out at the push of a button. It is worth mentioning some approaches do exist to automatically extract a model from some formal description of the system, like its source code, providing of course such description exists.

Several computation and modelling formalisms have been developed to express the many aspects in which a system can be described and analysed, which vary according to the study angle. Many of them take an automata-based approach, where the concept of *state* describes a “present situation of things” which evolves following some formally specified and thus unambiguous dynamics[†]. So from the current state s the automaton of the system can move to a next state s' following some *transition function* (or relation), which is typically denoted $s \rightarrow s'$.

In this state-based approach *nondeterminism* arises from the use of abstraction, e.g. when the system can be influenced by an unspecified environment, or several components run parallelly and only the global behaviour is of interest. Consider for instance a soda vending machine where the customer can request a *Cepsi* or a *Poke*. To simplify matters assume Gottfrid Svartholm and crew hacked the circuits so no payment is needed; the soda is obtained by pushing a button. In a model which abstracts away from the customer and considers the machine alone, there is no way to foretell which beverage will be chosen. So from a *standby* state there is a *nondeterministic choice* between a next *vend_Cepsi* state and a next *vend_Poke* state. A graphical depiction of this process is shown in Figure 2.1.

Just like in this toy example, nondeterminism can be described as a choice of the next state among a set of possibilities, viz. the transitions enabled on

[†] As stateless alternative see e.g. λ -calculus [Bar84] and the Haskell language.

each state are provided without any further information. Transition systems with labels is one of the most widespread formalism whose core purpose is to describe nondeterministic choices of the transitions between states.

Definition 1 (LTS). A finite *Labelled Transition System* (LTS) is a tuple $(S, s_0, A, \rightarrow, AP, Lab)$ where:

- $S \neq \emptyset$ is a finite set of *states*;
- $s_0 \in S$ is the *initial state* of the system;
- A is a finite set of *actions* or *labels*;
- $\rightarrow \subseteq S \times A \times S$ is the *transition relation*;
- $AP \neq \emptyset$ is a set of *atomic propositions*;
- $Lab: S \rightarrow 2^{AP}$ is a *labelling function*.

Having a single initial state and finite S and A sets suffices for the scope of this thesis, although Labelled Transition Systems can be defined in more general terms. See [BK08, Sec. 2.1] for a more complete introduction to these kinds of structures.

In Definition 1 the system transitions are defined by means of the \rightarrow relation. Element $(s, a, s') \in \rightarrow$ is denoted $s \xrightarrow{a} s'$. When such transition exists it is said that s' is a *successor* of s , and s is a *predecessor* of s' . Notice the labelling function Lab has domain on the states and is independent of the actions A . It relates a set $Lab(s) \subseteq AP$ of atomic propositions to state $s \in S$, which stands for the properties the state satisfies.

Figure 2.2 shows the soda vending machine modelled as an LTS. Each transition is decorated with an action, and for $AP = \{P, C, V\text{END}, \text{IDLE}\}$ each state was labelled according to the properties it should satisfy. For instance transition $\text{vend_Poke} \xrightarrow{\text{reset}} \text{standby}$ indicates standby is a successor of vend_Poke , and the placement of atomic proposition IDLE says the machine is idle only when standby is the current state of the LTS model.

The actions set A is provided without an ordering or any other information besides the set itself. In the vending machine example this means that when standby is the current state, there is no information a priori to indicate whether the system should evolve following the choose_P or the choose_C transition: the choice is nondeterministic. A closely related concept is *probability*. Depending on whether the underlying state space is discrete or continuous, the term *probabilistic choice* or *stochastic choice* is respectively

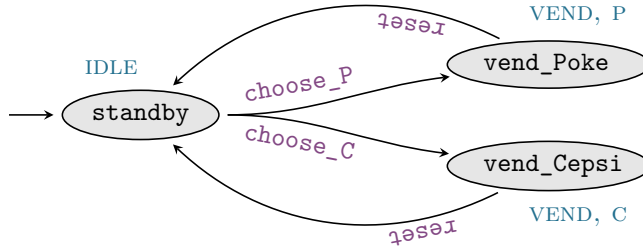


Figure 2.2: LTS of the soda vending machine

used to signify some quantification is provided for the transitions between states.

Probabilistic and stochastic behaviour can be naturally found in myriads of real-life situations, from the queueing in supermarkets to cloud formation and the failure and replacement of components in a cloud storage facility. In the discrete case, probability mass functions quantify the choice of the next state. For instance, if the successor states of s are s_1, s_2, s_3 with probability $1/2, 1/4, 1/4$ respectively, then observing $1 \ll N < \infty$ transitions from state s should result in, roughly, $N/2, N/4, N/4$ choices of state $s_1, s_2,$ and s_3 respectively. If we make *deadlocks* out of states $\{s_1, s_2, s_3\}$, i.e. add (only) the deterministic transitions $s_1 \rightarrow s_1, s_2 \rightarrow s_2,$ and $s_3 \rightarrow s_3$, these quantified transitions can be represented with the following *transition matrix*:

s	s_1	s_2	s_3
s	0	$1/2$	$1/4$
s_1	0	1	0
s_2	0	0	1
s_3	0	0	0

Here rows indicate the starting state of a transition, columns are the destination state, and the matrix elements are the probability of taking that transition. For instance, the probability of going from state s_1 (second row) to state s_3 (fourth column) is the matrix entry at $(2, 4)$, i.e. 0.

Markov chains are a well-known mathematical description of probabilistic behaviour. When the discrete notion of *step* rather than *continuous time* is inherent to the process evolution, the discrete-time variant of Markov chains can be used to model the system.

Definition 2 (DTMC). A finite *discrete time Markov chain* (DTMC) is a tuple $(S, s_0, \mathbf{P}, AP, Lab)$ where:

- $S, s_0, AP,$ and Lab are like in Definition 1 of LTS;
- $\mathbf{P}: S \times S \rightarrow [0, 1]$ is the *transition probability function* which for all $s \in S$ satisfies $\sum_{s' \in S} \mathbf{P}(s, s') = 1$.

System transitions are thus defined by means of \mathbf{P} , which is a formalisation of the transition matrix illustrated above. The value $\mathbf{P}(s, s') \in [0, 1]$ specifies for each state $s \in S$ the probability of taking the transition $s \rightarrow s'$. Such transition is said to exist iff $\mathbf{P}(s, s') > 0$, in which case s' is a successor of s and s is a predecessor of s' , very much like in the LTS case. The constraint imposed on \mathbf{P} ensures that each $\mathbf{P}(s, \cdot): S \rightarrow [0, 1]$ is a probability measure on S , see Appendix B. For a more profound study of DTMC see e.g. [Nor98, BK08].

To provide a concrete example consider a point-to-point socket connection through the Internet, with data-packets sent from one end and either lost or received at the other end. To study the proportion of successful transactions, the amount of packets sent and how many were received (rather than the time point at which this took place) provides all relevant information. Since the system evolves stepwise, where each step comprises either sending, receiving, or losing a packet, a Definition 2 can model the desired behaviour.

In a typical DTMC implementation the *transition probability matrix* is built, which explicitly gives the probability of moving from any state to the next at each step. This information suffices for systems where future behaviour depends exclusively on the current state and not on the path that led there. Furthermore the DTMC is assumed finite and time homogeneous, e.g. when a state is visited a second time the outgoing probabilities will be the same as they were the first time.

The matrix of transitions for states $\{s, s_1, s_2, s_3\}$ illustrated above is precisely a transition probability matrix. Notice that models with N states would need a N^2 square matrix of rational numbers. Efficient *abstract data types* exist to alleviate this necessity, such as sparse matrix representations or *multi-terminal binary decision diagrams* (MTBDD, [CFM⁺93, BFG⁺97]).

In comparison to the discrete world treated so far, the stochastic scenario is more involved because heed must be paid to measurability issues. Due to the memoryless property and the ease of analysis it offers, systems where all transitions are governed by the exponential distribution have been studied thoroughly. The continuous-time variant of Markov chains is the traditional choice to model these kind of systems.

Definition 3 (CTMC). A finite *continuous time Markov chain* (CTMC) is a tuple $(S, s_0, \mathbf{R}, AP, Lab)$ where:

- $S, s_0, AP,$ and Lab are like in Definition 1 of LTS;
- $\mathbf{R}: S \times S \rightarrow \mathbb{R}_{\geq 0}$ is the *transition rate function* which for all $s \in S$ satisfies $\mathbf{R}(s, s) = 0$.

In Definition 3 and as opposed to \mathbf{P} , matrix \mathbf{R} gives the exponential rates of taking transitions between states. Having $r = \mathbf{R}(s, s')$ means the probability of taking transition $s \rightarrow s'$ within t time units is $1 - e^{-rt}$. Notice that $r = 0$ yields a null probability, so as before the transition $s \rightarrow s'$ will be said to exist iff the matrix entry $\mathbf{R}(s, s') > 0$, with the corresponding definitions of predecessor and successor states. Also since the exponential distribution is memoryless, only knowing the current system state is enough to determine the future steps, just like for a DTMC.

Definition 3 allows multiple successor states, i.e. for any given state $s \in S$ there can be more than one other state $s' \in S$ s.t. $\mathbf{R}(s, s') > 0$. Such situations are known as *race conditions*. All outgoing transitions are enabled, and the first one to *fire* (according to a sampling of the corresponding exponential distributions) will be the one taking place. A deeper study of continuous-time Markov chains can be found in [Nor98, Bre68].

A classical CTMC example is the queueing e.g. at a supermarket cashier. The state space S would be used to represent the number of clients in the queue, whereas the rate at which new clients arrive and the cashier performs the service is encoded in \mathbf{R} . In a simple model this would result in a tridiagonal transition rate matrix, with null main diagonal.

So far the notions of nondeterministic and probabilistic/stochastic system evolution have been introduced. There is another key concept which naturally arises in many situations, whose explicit representation may be required. In fact this concept has already been mentioned though, until now, only incidentally: *time passage*.

A DTMC encodes the notion of discrete time evolution, whereas a CTMC deals, also implicitly, with continuous time. But what if the exact time point of occurrence of events needs to be modelled? For instance the soda vending machine may fall in a suspended state for 2 seconds after a Poke is chosen.

In general, consider systems in a continuous time setting where stochastically sampled events occur, yet not necessarily following the exponential distribution. A usual modelling solution is to associate a unique variable to each distinct event, which can sample time according to the desired probability

density function. Stochastic automata [DK05] follow this approach.

Definition 4 (SA). A finite *Stochastic Automaton* (SA) is a tuple $(S, s_0, A, \mathcal{C}, \rightarrow, AP, Lab)$ where:

- $S, s_0, A, AP,$ and Lab are like in Definition 1 of LTS;
- \mathcal{C} is a finite set of *clocks* such that each $c \in \mathcal{C}$ has an associated continuous probability measure $\mu_C: \mathbb{R} \rightarrow [0, 1]$ with support on $\mathbb{R}_{>0}$;
- $\rightarrow \subseteq S \times 2^{\mathcal{C}} \times A \times 2^{\mathcal{C}} \times S$ is the *transition relation*.

Clocks in SA explicitly mark the passage of time: each $c \in \mathcal{C}$ is assigned a positive value stochastically sampled from its associated probability density function μ_C , and decrease this value synchronously with the other clocks at constant speed, satisfying the differential equation $\dot{c} = -1$. When execution starts from s_0 , initial values for all clocks are sampled from their respective probability measures. Time can be considered to stop for a clock that has reached the value zero.

Even though the underlying notion of time is continuous, and just like with DTMC and CTMC, the succession of events in the execution of a stochastic automaton is discrete. More specifically, their occurrence is controlled by the *expiration* of the clocks. If the system is in state s and there is a transition $s \xrightarrow{C, a, C'} s'$, action $a \in A$ will be performed after all clocks in C have expired, viz. reached zero. Then the system moves to state s' sampling new values for the clocks in C' according to their probability measures. The *triggering clocks* of the transition are those in C , and the *resetting clocks* are those in C' . If $C = \emptyset$ or all triggering clocks have value zero when state s was reached, the transition can take place instantaneously.

Figure 2.3 shows a representation of the soda vending machine as a stochastic automaton. Clocks x_1 and x_2 have exponential probability density functions, with rates λ_1 and λ_2 respectively. Clock z instead samples its values from a continuous uniform probability measure with support in $[1.95, 2.05]$. Notice the state space is the same as in the LTS representation of Figure 2.2, but the nondeterministic choice of the transitions outgoing state `standby` was replaced with a stochastic one, by the inclusion of the triggering clocks x_1 and x_2 . Notice also the machine is suspended `VEND`-ing for roughly 2 time units when a `Poke` is chosen, which was modelled in the transition going from `vend_Poke` to `standby` by using z as a triggering clock.

We highlight SA include nondeterministic behaviour, unlike DTMC and CTMC. That is clear from the definition of the transition relation \rightarrow : from

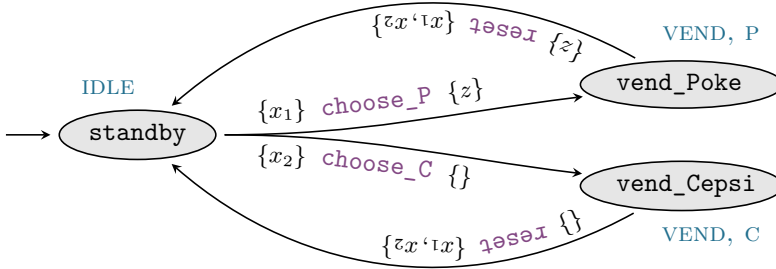


Figure 2.3: SA of the soda vending machine

any state s there can be several transitions with the same triggering clocks C , resetting clocks C' , and action label a , reaching different target states s' . In later chapters a restricted version of Definition 4 will be introduced, which rules out nondeterminism by construction. Namely, by imposing restrictions in the nature of relation \rightarrow , only stochastic behaviour can be expressed.

The formalisms and examples presented so far were chosen as simple as possible with the intention of introducing the fundamental concepts of *nondeterminism*, *probability/stochasticity*, and *time evolution*, in a manner directly applicable to the needs of this thesis. One could however be interested in modelling more general systems, where time elapses at different speeds for some components, or probabilistic and nondeterministic behaviour are intertwined in a discrete setting.

Many more formalisms exist to satisfy each particular need. For instance *Markov Decision Processes* (MDP, [Bel57]) mix the nondeterminism from LTS with the probabilistic transitions of DTMC, and *Stochastic Hybrid Automata* (SHA, [FHH⁺11]) allow the description of non-homogeneous time passage. The interested reader is referred to [HHHK13, Table 3] for a broader overview of the options available in the literature. Furthermore the lattice presented in [Har15, Figure 1.2] summarises the relationships between several modelling formalisms, in terms of the behavioural concepts they can express.

2.2 Model property queries

The intricate task of distilling a model from a real system is not done for the mere pleasure of it, but to gain some insight and study the properties the (model of the) system exhibits. Ensuring all desired requirements are met

is usually the main intention. It may also be possible to obtain some useful performance measurements from the model. All this is attained by *querying* the model.

Coarsely one can speak of *qualitative* vs. *quantitative* queries: qualitative (or functional) queries deal with absolutes such as termination or functional correctness; quantitative queries are used to study performance and efficiency aspects such as power consumption or time to termination.

Consider an ICE train service between Hamburg Hbf[†] and Köln Hbf, with a single intermediate stop at Hannover Hbf. In this example qualitative questions are “*could the train be overcrowded, having to leave some passengers at Hamburg Hbf?*” and “*does a departing train always reach destination?*” Quantitative questions are “*what is the average amount of passengers in a train trip?*”, “*how likely is it for an incautious commuter to find a full train at Hannover Hbf station and lose the trip?*” and “*do more than 99% of trains trips reach destination safely?*”

Just like systems can be specified in some formal language, so do properties. Logics are developed to describe the desired properties using propositions produced by their grammar. *Temporal logics* are a usual choice due to their expressiveness regarding execution paths: they produce succinct descriptions of possible executions of the system, i.e. of the possible successions of states the formal model allows (henceforth *paths*). Notice paths need not be finite.

Using propositions derived from the grammar of a temporal logic, concepts relevant for real-life systems like *reachability* (“is this situation feasible?”), *safety* (“something—bad—never happens”), and *liveness* (“something—good—will always eventually happen”), can be compactly and clearly stated.

Many alternatives are available when searching for the proper logic, the best choice depending on the behaviour to describe. Popular options are listed and commented on next. No formal definition of their syntax nor their semantics is provided; instead, to give a hint of how these logics talk about system execution, a few minimal examples are presented in each case.

Linear Temporal Logic (LTL, [Pnu77]) allows reasoning about a single time line, viz. no transitions branching is considered and thus there is a single successor to each state in a path. LTL offers the disjunction (\vee) and negation (\neg) propositional logic operators, and the *next* (\circ) and *until* (U) temporal modal operators. The usual propositional operators can be derived from $\{\vee, \neg\}$; some relevant derived temporal modal operators are *eventually* (\diamond), and *always* (\square).

[†] Hauptbahnhof, aka main train station.

LTL formulae have semantics in the system paths, and for Definitions 1 to 4 they describe these paths by referring to the atomic propositions they will visit. In a nutshell, \bigcirc talks about “the (single) next state of this path,” \diamond means “some state further ahead in the path” (viz. in the future), \square means “all future states including the current one,” and \bigcup says “something happens in all states of the path until something else finally happens.” Some simple LTL formulae are:

- $\bigcirc \text{VEND}$ “*next thing to happen is a customer choosing a beverage,*” so in the LTS of the vending machine from Figure 2.2, this formula is true on any state of a path located one transition away from a **VEND**-labelled state, which is satisfied only by **standby**;
- $\diamond(\text{VEND} \wedge \text{P})$ “*eventually a Poke will be chosen,*” true when some state ahead in the path is labelled with both **VEND** and **P** (i.e. when **vend_Poke** is reachable);
- $\square \neg \text{OVERCROWD}$ “*the train is currently not overcrowded and it will never be,*” so in the ICE train case this query is satisfied by states in paths ahead of which no **OVERCROWD**-labelled states are visited;
- $\neg \text{OVERCROWD} \bigcup \text{KÖLN}$ “*the train will not be overcrowded until it finally arrives at Köln Hbf,*” where the eventual arrival at Köln is necessary to satisfy the property.

Computational Tree Logic (CTL, [CE81]) considers a branching time scenario, where there are many possible successors to every state and all of these are to be quantified upon. In contrast to LTL this demands a state-based semantics, since all paths (instead of a single one) rooted in the current state are considered. Besides the propositional and temporal operators from LTL, CTL offers the *for all paths* (\forall) and *for some path* (\exists) operators. The first is satisfied if *all* paths originating from the current state satisfy the rest of the formula (which must start with a temporal operator: \bigcirc , \bigcup , ...); the second is satisfied if *some* path satisfies the rest of the formula. The following are CTL formulae:

- $\exists \diamond \text{VEND}$ “*there is a system execution starting at the current state, where eventually a customer will choose some beverage,*” which is a tautology easy to verify;
- $\forall \diamond(\text{VEND} \wedge \text{P})$ “*all system executions from the current state will eventually see a customer choosing a Poke,*” which is false in **standby**

since there is a (strange but valid) path where only Cepsis are chosen:

$$\text{standby} \xrightarrow{\text{choose_C}} \text{vend_Cepsi} \xrightarrow{\text{reset}} \text{standby} \xrightarrow{\text{choose_C}} \dots \quad (1)$$

It is worth mentioning that even though LTL and CTL formulae can sometimes be encoded in terms of each other, in general the expressiveness of these logics is incomparable [BK08, Theo. 6.21]. For instance the CTL formula $\forall \diamond \forall \square a$ cannot be expressed in LTL, whereas the LTL formula $\diamond \square a$ cannot be expressed in CTL.

Probabilistic Computational Tree Logic (PCTL, [HJ94]) considers also the probabilistic nature of the Markov chains it was designed to study. Whereas LTL and CTL talk about which states to visit and which not, PCTL allows the user to express the probability of following certain path. This logic is based on CTL with a major difference: the existential and universal quantifiers are replaced with a probabilistic operator (\mathbf{P}), of which \forall and \exists could be considered special cases (even though that is not strictly true). The formula $\mathbf{P}_I(\phi)$, where I is a rational-bounded interval of $[0,1]$ and ϕ is a path, asks whether the probability of executing ϕ is within I :

- $\mathbf{P}_{\geq 0.9}(\neg \text{OVERCROWD} \mathbf{U} \text{HANNOVER})$ “the probability of having the ICE train overcrowded in the first trajectory Hamburg–Hannover is below 10%,” providing the train does effectively reach Hannover;
- $\mathbf{P}_{\leq 0.25}(\diamond \text{OVERCROWD})$ “at most one train from every four can get overcrowded,” which applies to any stage of the train trip;
- $\mathbf{P}_{>0}(\square (\text{IDLE} \vee \mathbf{P}))$ “there is a chance of choosing only Pokes”;
- $\mathbf{P}_{=1}(\diamond (\text{VEND} \wedge \mathbf{P}))$ “eventually a Poke is chosen, almost for sure.”

Remarkably, the last example is *not equivalent* to formula $\forall \diamond (\text{VEND} \wedge \mathbf{P})$ from CTL. There may be paths with zero probability (see Appendix B) where $\diamond (\text{VEND} \wedge \mathbf{P})$ is false, which are disregarded by $\mathbf{P}_{=1}$ in PCTL but not by \forall in CTL—one such path was given in eq. (1). So $\forall \diamond (\text{VEND} \wedge \mathbf{P})$ is false in the LTS model from Figure 2.2. On the other hand, in the SA model of Figure 2.3 where probabilities come into play, $\mathbf{P}_{=1}(\diamond (\text{VEND} \wedge \mathbf{P}))$ evaluates to true, since the probability of choosing paths like in eq. (1) is zero.

Something similar happens between the CTL formula $\exists \square (\text{VEND} \wedge \mathbf{P})$ and its probabilistic counterpart $\mathbf{P}_{>0}(\square (\text{VEND} \wedge \mathbf{P}))$. In general the expressiveness of CTL and PCTL are incomparable [BK08, Lemmas 10.44 and 10.45].

Continuous Stochastic Logic (CSL, [BKH99]) is yet another branching-time temporal logic specifically designed for CTMC analysis. It is based on CTL and similar to PCTL, with the additions of a time bounded version of the until temporal operator ($U^{\leq t}$), also extended to its derivatives (e.g. $\diamond^{\leq t}$), and an operator to reason about the steady-state probabilities of the system (S). The bounded versions of the temporal operators limit the time horizon to consider, so $\diamond^{\leq t}$ means sometime within t time units. The S operator considers instead an infinite time horizon, talking about probabilities of paths in a system in *equilibrium*. The following are CSL formulae:

- $P_{\geq 1}(\neg \text{OVERCROWD } U^{\leq 111} \text{ KÖLN})$ “(almost surely) the train will arrive at Köln within 111 time units, and it will not become overcrowded during the whole trip”;
- $P_{[.5,.6]}(\diamond^{\leq \frac{3}{2}} (\text{VEND} \wedge P))$ “within one and a half time units, the probability of a customer choosing a Poke is between 50% and 60%”;
- $S_{\geq 0.9}(\Box^{< 60} \neg \text{OVERCROWD})$ “during a normal working day (aka in equilibrium) and with at least 90% probability, the ICE train will never become overcrowded in the first 60 time units.”

So far only Boolean-valued formulae have been considered: even in quantitative queries like $P_{\leq 0.25}(\diamond \text{OVERCROWD})$ the answer is either “yes” or “no” for a given path ϕ . The *performance itself* can also be the query, e.g. “what is the probability of this happening?” during the execution of ϕ , rather than the requirement “is such probability lower than certain value?”

In general quantitative questions like these, which request the actual value of an efficiency measure, are omitted in temporal logics. The problem is that the nesting of numeric-valued answers is hard to grasp in a consistent manner. Several tools however offer the user the possibility to perform such queries at the top level of their formulae, of which the operators $P=?$ and $S=?$ in PRISM are modern examples [KNP07, Sec. 5.1].

2.3 Analysis of the model

Once the system model and queries have been formally specified, automatic algorithms can be run to check whether the model satisfies the requirements or to find out the performance of (the formalisation of) the system. There is more than one way of doing this; a few popular techniques are introduced next, explaining in greater depth the one relevant for this thesis.

2.3.1 Overview and known approaches

When thinking of automatic formal verifications, *automated theorem proving* (ATP, aka automatic deduction) may be the first method to come to mind. It comprises using computer programs to show that some statement (the *conjecture*) is a logical consequence of a set of statements (the *axioms* and *hypotheses*). This of course requires an appropriate formulation of the problem as axioms, hypotheses, and a conjecture.

ATP is a broad term mostly associated to proof assistants. A *proof assistant* is a software tool which helps developing formal mathematical proofs by means of human-machine collaboration. This technique is not completely automatable and in any case falls outside the scope of this thesis. Curious readers are referred to e.g. the Coq proof assistant [dt04].

More in line with the modelling viewpoint introduced so far, *model checking* is a first-choice strategy with roots in the exploration of the state space of a formal model of the system under study. Graph analysis, numeric approximations, fixed-point estimations, and several other computer-based methods are covered by this umbrella term.

In its various forms model checking can express and study nondeterministic, probabilistic/stochastic, and timed systems. The general idea is having automatic checks run on the model, where the property under study specifies what to look for and thus which algorithm to execute. For qualitative queries either the property is proved to hold or (usually) a counterexample path is given as output. For quantitative queries involving iterative procedures many implementations choose, or require the user to input, a *convergence epsilon*, and terminate computation as soon as the difference between the outcome of two consecutive iterations is less than this value.

Whichever its flavour, the core algorithmic set used by model checking requires a representation of all states of the model. This leads to the infamous *state explosion problem*, since the size of the state space grows exponentially with the number of variables in variable-based formalisms, which are the input languages of most modern model checking tools—see PRISM [KNP11], UPPAAL [BDL⁺06], MODEST [HHHK13], STORM [DJKV16], etc.

Many techniques exist to reduce, truncate, or abstract the state space without affecting the model checking results, or affecting them in a known and quantifiable manner. This has enabled the study of several real-life systems with very large state spaces. Yet the problem is inherent to model checking and state space reduction is an active area of research.

A different approach to analyse models, which is theoretically oblivious of

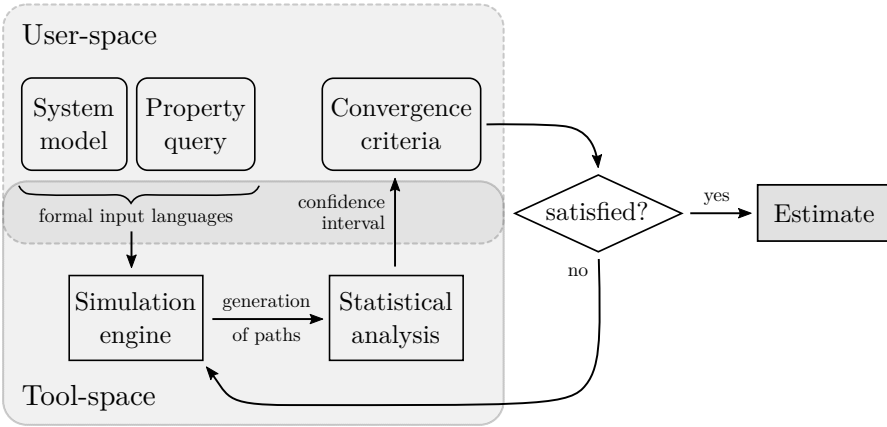


Figure 2.4: Analysis by Monte Carlo simulation

the number of system states, is *Monte Carlo simulation* or merely *simulation*. In its standard form, model analysis by simulation comprises the generation of paths following the formal description of the system. These paths need to be finite, so either the model naturally expresses finite executions or some termination or truncation criteria has to be forced upon the generation procedure. Paths are then examined from the viewpoint of the query to determine whether or how the property is satisfied.

When the simulation mechanism is properly implemented, each new path provides fresh, independent information about the satisfiability of the property by the model. By means of some statistical analysis this is deemed sufficient at certain point, after *enough paths have been generated*. By then an estimate of the answer to the query is available for the user. See Figure 2.4 for a schematic representation of this procedure.

The question then arises, what does *enough simulation*—viz. generation of paths—mean? Computation in model checking stops either when the whole state space has been analysed, a state sought has been reached, or the iterative procedure converges up to the epsilon imposed. These concepts do not apply to the approach of analysis by simulation. Instead a statistical notion of convergence derived from the law of large numbers is used to judge how far the current estimate is from the real answer. It is up to the user to choose the desired proximity, which is usually done in terms of confidence criteria, as it will be explained in more detail in Section 2.3.4.

This thesis concerns itself with the automation of a particular approach

to analyse models by simulation, in a scenario where the nature of either the model or the query make it very unlikely to generate useful paths. In such scenario standard simulation techniques are rendered useless, due to the rarity of the event which needs to be observed for the statistical analysis to converge. This is usually referred to as *rare event simulation* (RES, [RT09a]), and implies the use of intelligent techniques to speed up convergence.

A few remarks are due before concluding this section. Recall from Section 2.1 the three axes for model characterization introduced: nondeterminism, probability/stochasticity, and (explicit) time. On the one hand, the latter two can be easily encoded in simulation. Stochastic models even constitute an ideal field for applying this technique, since formal analysis and numerical approximations can be hard to develop for general and involved cases. Instead *discrete event simulation* offers a straightforward solution which is relatively easy to implement.

On the other hand, when faced with a nondeterministic choice and by the very definition of it, a simulation does not know which way to go. This poses no problem for model checking which can simply branch and follow all choices simultaneously. Path simulation however finds here a limiting factor, though some efforts are being currently made in this direction. Namely, [BFHH11] shows a way to get rid of spurious nondeterminism, which is certainly no final solution. Simulation of *true* or non-spurious nondeterminism is dealt with in [HMZ⁺12, BCC⁺14] with some issues, like requiring well-structured problems and showing bad performance in scenarios where optimal scheduling decisions are needed. The theory from [LST14] works on MDP models, encoding schedulers implicitly which saves on memory consumption. Among the most modern contributions stand [DLST15, DHLS16], the latter working on Probabilistic Timed Automata (which generalise MDP with clock variables) using a “lightweight approach.”

Finally it is noted that, when embedded within the setting of formal model description and verification, the simulation approach has often received the name *statistical model checking*. This trend is not followed in the thesis since the formal guarantees ensured for the results of model checking are incomparable to the statistical notions of confidence and precision inherent to the Monte Carlo approach. This in spite of partial overlapping of the problems these two techniques can solve. Furthermore there lies the issue of nondeterminism, which as explained can be naturally dealt with by model checking yet not by simulation. Thus and henceforth the term *simulation* will be used for the technique of model analysis by the generation and statistical analysis of system paths.

2.3.2 Simulation

There exist at least two different approaches to simulate paths from a system model specification. In *continuous simulation* the succession of relevant events is assumed to evolve continuously. This is typically the case for differential equations that give relationships for the rates of change of the state variables with time. For concrete examples think of rocket trajectory tracking and simulation on FPGA circuits. Numerical analysis is often applied in such situations, e.g. Runge-Kutta integration. Another implementation consists in discretising time into small enough slices and sequentially see to all activity taking place at each slice.

Discrete event simulation (DES, [LK00]) offers an alternative best suited for systems that naturally evolve at discrete time points. No regularity in time is required, i.e. these time points need not be equally spaced. What matters is that system evolution takes place stepwise, so the simulator can build a list of future events which will be dealt with orderly, one after another. The correspondence between this approach and the automata-based model description discussed in Section 2.1 is evident: DES is the usual way to perform model analysis by simulation in such formalisms.

Even though the concrete implementations may vary, the basic ingredients in DES can be always identified as follows:

- *State*:
 - ▷ at any specific time point all system components have a clearly defined and unambiguous *state*;
 - ▷ states convey the notion of “current situation of things,” e.g. number of customers in the queue, number of failed disks in a cluster, busy/free repairman module, etc;
 - ▷ the state of all components regarded en masse is the *system state*.
- *Event*:
 - ▷ an *event* is an instantaneous change in the system state;
 - ▷ not all system components must be affected by an event; a single one changing its state is sufficient;
 - ▷ states *only* change during the occurrence and handling of events;
 - ▷ an event should be *atomic*: even though the low-level implementation may manage the state change of some components before others, this shall not affect the overall final outcome at global scope;

- ▷ from a set of possible events, usually the current system state defines which are enabled and which not.
- *Prioritised list* :
 - ▷ during simulation, future events are scheduled and ordered according to some priority, e.g. occurrence in time;
 - ▷ all these future events are stored in some abstract data type, like a list or queue, where they are kept for later handling;
 - ▷ the next most important event can be efficiently obtained from such list, usually in constant time.
- *Random numbers generation* :
 - ▷ DES is customarily employed in probabilistic/stochastic cases, which require some way to randomise the generation of events;
 - ▷ usually a pseudo- or quasi-*random number generator* is employed as the seed of all randomness;
 - ▷ several techniques are known to transform the $[0, 1]$ -ranged value of the random number generator into probabilistic/stochastic behaviour following the desired distributions [PTVF07].

Assuming the presence of all the components mentioned above, a high-level application of DES can be described as follows:

1. Setup the initial system state.
2. Based on the initial configuration resulting from item 1 plus the description of the system, generate a set of initially enabled events and orderly store them in the prioritised list of events (the *events list*).
3. Fetch *the* next event, according to the order of the events list.
4. Modify the system state following the event resulting from item 3.
5. The new system state resulting from item 4 may enable new events and disable old ones. The events list must be updated accordingly, making sure the resulting outcome respects the events priority criteria.
6. Go back to item 3.

The iterative procedure described above finishes either when the event list empties, or some user-defined condition for ending the simulation is arrived at, e.g. reaching N simulated time units.

Once DES stops, the relevant gathered data is fed to the statistical analysis mechanism as a *fresh sample*. The estimate answer to the user query is then updated and termination is considered: if enough samples have been collected to ensure the desired statistical confidence, no more simulation is needed. Otherwise another simulation is started.

There are also scenarios where the data for statistical analysis can be gathered during simulation itself, viz. no termination of DES is strictly needed. Instead, statistical information can be obtained and analysed during the iterative procedure, which is for instance the case when asked for the average size of a queue, or the steady-state availability of a resilient system.

2.3.3 Estimation

So far the existence of statistical procedures to process the data obtained from simulation has been mentioned yet not explained. The required notions on statistics are introduced here, both because it is the following obvious subject to cover, and also since it will help to restate in a more formal sense one of the motivations of this thesis. Some basic knowledge is assumed regarding the theory of statistics—see e.g. [Ric06] for a reference on the subject.

The Oxford Dictionary of English defines *statistics* as the science of collecting and analysing numerical data in large quantities with some inference purpose [Oxf13]. For the scope of this thesis all numerical data will come from discrete event simulation executed automatically in a computer. Each individual value will be called *sample* and denoted X or X_i . Each sample will be the result of a simulation and can be interpreted as the outcome of some *random variable* (r.v.) whose distribution is not necessarily known.

The sampling distribution is inherent to the nature of the system model. It should be clear that, assuming a correct use of the underlying random number generator [PTVF07], the samples generated can be considered pairwise independent and identically distributed (IID) in the statistical sense.

Sampling will be the process of executing discrete event simulation to generate IID samples. The data sequence resulting from sampling will be called a *random sample* and denoted $\{X_i\}_{i=1}^N$, signifying N IID samples were generated in the order X_1, X_2, \dots, X_N , where N is called the *sample size*. From the mathematical point of view $\{X_i\}_{i=1}^N$ can be seen as a sequence of IID random variables.

Given thus some random sample consider the average of its values, which will be denoted \bar{X}_N for size N :

$$\bar{X}_N \doteq \frac{1}{N} \sum_{i=1}^N X_i. \quad (2)$$

This value will be called the *sample mean* and is mostly used as an estimator of the *population mean* [SS07]. Since it is the transformation of random variables, the sample mean is a random variable itself, with its own distribution, mean, variance, and other statistical moments. The following results will help to study the expressions of these parameters, to gain some insight on how they shape the RES problem. The proofs are not included; they can be found in most textbooks on (mathematical) statistics, e.g. [Bil12,Ric06].

Proposition 1 (Chebyshev's inequality). *Let X be a random variable with mean μ and variance σ^2 . Then for any real number $\varepsilon > 0$*

$$P(|X - \mu| > \varepsilon) \leq \frac{\sigma^2}{\varepsilon^2}.$$

Roughly speaking Proposition 1 suggests that for small enough σ^2 the chances are high that an outcome of the r.v. X comes close to the mean $E(X) = \mu$. This is in line with the idea that the standard deviation of a random variable indicates how spread out its possible values are.

It can be proven from eq. (2) that the sample mean is in fact an *unbiased estimator* of the population mean, viz. $E(\bar{X}_N) = \mu$ where μ is the (usually unknown) mean of the random variables $\{X_i\}_{i=1}^N$. Proposition 1 then says that \bar{X}_N could be made arbitrarily close to μ , providing some way to reduce its variance was known and applicable. In turn one has the intuitive idea that the bigger the random sample, the more accurately the sample mean will resemble μ , i.e. the less such average will deviate from the real mean. This suggests that N and σ^2 may be inversely proportional magnitudes, which turns out to be precisely the case given the independence of the X_i :

$$\text{Var}(\bar{X}_N) = \frac{\sigma^2}{N}. \quad (3)$$

Equation (3) tells then how to reduce the variance in the formulation of Chebyshev's inequality: substituting X for \bar{X}_N and increasing the sample size N should indeed make \bar{X}_N closer to μ . The following result, which is a consequence of Proposition 1 and eq. (3), formalises a generalisation of this statement.

Theorem 2 (Law of Large Numbers). *Let $X_1, X_2, \dots, X_i, \dots$ be a sequence of independent random variables with $E(X_i) = \mu$ and $\text{Var}(X_i) = \sigma^2$ for all $i \geq 1$. Let \bar{X}_N be the average up to the N -th random variable, viz. $\bar{X}_N \doteq \frac{1}{N} \sum_{i=1}^N X_i$. Then for any real number $\varepsilon > 0$*

$$P\left(|\bar{X}_N - \mu| > \varepsilon\right) \rightarrow 0 \quad \text{as } N \rightarrow \infty.$$

Notice Theorem 2 does not require the random variables to be identically distributed. The strictly weaker condition of having the same first and second moments is enough to ensure the result, which is also clearly satisfied by the simulation approach studied.

In the proposed setting where sampling comes from discrete event simulation, Theorem 2 can be interpreted as follows: when estimating the average behaviour of the model, several executions should be simulated; the more paths one generates, the closer their mean can be expected to resemble the true average behaviour.

Furthermore the resemblance can be made arbitrarily close, as long as one can keep producing samples. Conceptually, the estimation of the average behaviour will be improved by increasing the sample size. This is proved in [CR65] assuming that the random sample converges and $\sigma^2 < \infty$.

2.3.4 Convergence and stopping criteria

The Law of Large Numbers provides sufficient conditions for the informal notion of *enough sampling* to have a meaning, where one can choose a priori a desired proximity to μ and then produce enough samples until it is achieved. However this result only states such value exists, i.e. that there is an N which will satisfy the user's needs[†]. It speaks nothing about how to effectively measure the proximity to μ of the current sample mean.

Quantifying this proximity is of great practical use. Typically when analysing a model with simulation, the user requests the estimation of some value within certain accuracy. Independent simulations are launched to generate the random sample, from which e.g. the sample mean is used to build an estimate for the value sought. The Law of Large Numbers says a time will come when enough samples have been generated to grant the requested accuracy, yet that is not enough for practical purposes. It is crucial to know, or give some guarantees about, how close the current estimate is

[†] Strictly speaking this is stated in the *strong law of large numbers*, a variant of Theorem 2.

to the real value, and whether more samples should be generated. Some termination criteria is of the essence.

Luckily there are limiting properties of the sum of random variables which provide ways to measure the accuracy of the current estimate, helping to build a termination criterion. Notice that for any random variable X the *standardised random variable* Z defined as

$$Z \doteq \frac{X - E(X)}{\sqrt{\text{Var}(X)}}$$

satisfies $E(Z) = 0$, $\text{Var}(Z) = 1$. Let C_N be the sum of some random sample of size N with mean μ and variance σ^2 , i.e. $C_N \doteq \sum_{i=1}^N X_i$. Theorem 2 says C_N/N converges to μ ; the following result studies this convergence and gives an explicit cumulative distribution function for the standardization $Z_N \doteq (C_N - N\mu)/(\sigma\sqrt{N})$.

Theorem 3 (Central Limit Theorem). *Let $\{X_i\}_{i=1}^N$ be a random sample where all r.v. have mean μ and finite positive variance σ^2 . Let $C_N = \sum_{i=1}^N X_i$, then*

$$\lim_{N \rightarrow \infty} P\left(\frac{C_N - N\mu}{\sigma\sqrt{N}} \leq z\right) = \Phi(z)$$

for finite $z \in \mathbb{R}$, where $\Phi(z)$ is the cumulative distribution function of the standard normal distribution:

$$\Phi(z) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^z e^{-\frac{x^2}{2}} dx .$$

Theorem 3 talks about convergence in distribution, stating the standardised sample mean Z_N converges to the cumulative distribution function of the standard normal distribution. Generally speaking the *speed of convergence* depends on the real distribution of the X_i , high skewness and long tails playing against it. Several rules of thumb exists about which N is large enough to start using the approximation of the Central Limit Theorem, e.g. $N > 30$, or $C_N > 5 \wedge N * (1 - C_N/N) > 5$ for binomial proportions. Of course and in general, the larger the sample the better the approximation.

Strictly speaking Theorem 3 should be applied if the variance of the population is known. In most practical situations σ^2 is unknown and approximated with the unbiased estimator

$$S_N^2 \doteq \frac{1}{N-1} \sum_{i=1}^N (X_i - \bar{X}_N)^2 . \quad (4)$$

In such cases the longer-tailed *Student's t-distribution* with $N - 1$ degrees of freedom should be used instead, since it does not depend on the population value σ^2 . Formally:

$$\frac{\bar{X}_N - \mu}{S_N/\sqrt{N}} \sim T_{N-1} \quad (5)$$

where $S_N \doteq \sqrt{S_N^2}$, μ is unknown, and the Student's t-distribution with $\nu \in \mathbb{R}$ degrees of freedom is characterised by the probability density function

$$f_\nu(t) = \frac{1}{\sqrt{\nu} B(\frac{1}{2}, \frac{\nu}{2})} \left(1 + \frac{t^2}{\nu}\right)^{-\frac{\nu+1}{2}}$$

where B is the Beta function. The corresponding cumulative distribution function $T_\nu(t)$ is harder to express and thus not included. It is a known result that $T_\nu(x)$ converges to $\Phi(x)$ when $\nu \rightarrow \infty$, coherently relating eq. (5) with Theorem 3.

From the practical point of view and given the symmetry of the cumulative distribution function of the Student's t-distribution (i.e. $T_\nu(-t) = 1 - T_\nu(t)$), this means that for sufficiently large N one can assume that $P(-t < Z_N < t) \approx 2T_{N-1}(t) - 1$. Notice the use of the standardization

$$Z_N = \frac{\bar{X}_N - \mu}{\sigma/\sqrt{N}}$$

since $C_N = N \bar{X}_N$, where S_N from eq. (4) is to be used in the above equation as an approximation for σ when the population variance is unknown.

To see how these results fit in the scenario of model analysis by simulation, suppose the user wants to find out the likelihood γ of satisfying certain property in the model. Furthermore, and here lies the core asset, he requests an upper bound of $\varepsilon > 0$ for the probability of error in the estimation.

The standard Monte Carlo approach via discrete event simulation generates several, N say, independent simulations. Each simulation results in some path which will either satisfy the property or not. Thus a random sample $\{X_i\}_{i=1}^N$ is generated, where $X_i = 1$ if the i -th simulated path satisfies the property and $X_i = 0$ otherwise. This definition of the X_i means the queried likelihood is the population mean, $\gamma = \mu$. Thus a straightforward estimator $\hat{\gamma}$ for the likelihood is the sample mean \bar{X}_N . Denote $\bar{X} = \bar{X}_N$ and $\sigma_{\bar{X}} = S_N/\sqrt{N}$, then the following yields a (conservative) quantification of the

error incurred in the approximation:

$$\begin{aligned} P(|\hat{\gamma} - \gamma| \leq \varepsilon) &= P(-\varepsilon \leq \hat{\gamma} - \gamma \leq \varepsilon) \\ &= P\left(-\frac{\varepsilon}{\sigma_{\bar{X}}} \leq \frac{\bar{X} - \mu}{\sigma_{\bar{X}}} \leq \frac{\varepsilon}{\sigma_{\bar{X}}}\right) \\ &\approx 2T_{N-1}\left(\frac{\varepsilon}{\sigma_{\bar{X}}}\right) - 1. \end{aligned}$$

Notice only *point estimates* have been considered so far, i.e. the user is given an estimate $\hat{\gamma} \in \mathbb{R}$ of the real value γ he wishes to know, and can compute the probability of error incurred in the estimation. The approach usually followed in practice is slightly more involved and adds an interval to the information provided by the point estimate.

Definition 5 (Confidence Interval). Given some random sample $\{X_i\}_{i=1}^N$ drawn from a population, a *confidence interval* (CI) around some parameter $\theta \in \mathbb{R}$ of the population is an interval $[l, u] \subset \mathbb{R}$, whose bounds l, u are random variables derived from the sample, and which contains (*covers*) the real parameter θ with some known probability.

Definition 5 is rather lax because the specific expression of the interval may vary depending on the parameter to estimate and the nature of the sample. For this thesis the main interest is to build a CI around the population mean μ . Denote by z_α the α -quantile of the standard normal distribution for $0 < \alpha < 1$, i.e. the area to the right of $z_\alpha \in \mathbb{R}$ under the curve of its density function is α . Using the symmetry of this function together with the approximation provided by the Central Limit Theorem this means

$$P\left(-z_{\frac{\alpha}{2}} \leq \frac{\bar{X} - \mu}{\sigma_{\bar{X}}} \leq z_{\frac{\alpha}{2}}\right) \approx 1 - \alpha$$

or equivalently

$$P\left(\bar{X} - z_{\frac{\alpha}{2}}\sigma_{\bar{X}} \leq \mu \leq \bar{X} + z_{\frac{\alpha}{2}}\sigma_{\bar{X}}\right) \approx 1 - \alpha. \quad (6)$$

Equation (6) expresses clearly that for sufficiently large samples, the probability that μ lies in the interval $\bar{X} \pm z_{\frac{\alpha}{2}}\sigma_{\bar{X}}$ is approximated by $1 - \alpha$. This interval is thus called a $100(1 - \alpha)\%$ *confidence interval*. The value $C_L = 100(1 - \alpha) \in (0, 100)$ is the *confidence level* and its width $2z_{\frac{\alpha}{2}}\sigma_{\bar{X}}$ is the *precision* of the interval. The same analysis follows using Student's t-distribution instead of the normal distribution, when the population variance

is unknown. The *theoretical coverage* for a given confidence level C_L states that, out of $M \gg 1$ intervals of confidence level C_L generated from IID samples, $M \frac{C_L}{100}$ of them should contain the real value μ .

All ingredients are now ready to perform a full model analysis by simulation. The user provides a formal description of the model and the property to verify in some temporal logic. He also specifies the desired confidence level and precision for the estimation. Simulations are sequentially run using the discrete event simulation approach, yielding concrete values for \bar{X} and $\sigma_{\bar{X}}$ (or whichever estimate $\hat{\gamma}$ and its variance correspond). Since the confidence level has been fixed, this yields in turn concrete values for the precision of the interval, which according to Theorem 2 will eventually decrease. Computation stops as soon as the achieved precision falls below the one requested, i.e. when our estimate is *accurate enough*. As an alternative approach the user could choose a confidence level and simulation time, and measure the achieved precision once simulation finishes.

2.4 Rare events

The recipe provided in Sections 2.3.2 to 2.3.4 to estimate an answer for the user query by simulation is fairly broad. Theoretically it is only limited by the feasibility to generate simulation paths on the model (usually a straightforward task) and the computability of the expression of the estimator used. It is nevertheless its efficiency, rather than its generality, what limits its application.

Denote by $z \in \mathbb{R}$ the quantile from eq. (6), regardless of whether a standard normal distribution or Student's t-distribution is used. Also, denote by $\hat{\sigma}$ the measured standard deviation of the estimator $\hat{\gamma}$ used to approximate the value of the real unknown value γ . The speed of convergence of the iterative simulate/estimate-procedure is strongly related to the precision requested for the CI, i.e. $2z\hat{\sigma}$. For instance when the estimator is the sample mean, $\hat{\gamma} = \bar{X}_N$, the precision decreases as the inverse square root of the sample size, since then either $\hat{\sigma} = \sigma/\sqrt{N}$ or $\hat{\sigma} = \sqrt{S_N^2/N}$. This *convergence speed* is known to be moderately efficient in many practical applications.

Nonetheless and as earlier stated, this thesis is concerned with the study of rare events, meaning $0 < \gamma \ll 1$. For very small numbers, e.g. $\gamma \approx 10^{-8}$ and smaller, the *absolute error* given by the expression $z\hat{\sigma}$ is not representative enough [RT09a]. Instead the *relative error* captures in a more flexible and meaningful way the accuracy of the estimation.

Definition 6 (Relative Error). Let $[l, u]$ be a confidence interval built around some parameter γ with precision $2z\hat{\sigma}$. The *relative error* (RE) of the confidence interval is its absolute error divided by the parameter:

$$\text{RE}_{[l,u]} \doteq \frac{z\hat{\sigma}}{\gamma}.$$

Usually the real value of γ is unknown, in which case the estimate $\hat{\gamma}$ is to be used for sufficiently large samples. The relative error can be thought of in terms of the precision of the interval relative to the estimated value. Asking for a 10% relative error means e.g. computation will stop when the half-width of the CI built around the estimate is smaller or equal than $\hat{\gamma}/10$, i.e. 10% of the estimate.

Figure 2.5 shows graphically why using relative error when dealing with rare events is important and more flexible than working with absolute errors. Since one does not know a priori the magnitude of γ , requesting for instance 10^{-7} of interval precision might seem tight enough. Yet if γ is even one order of magnitude lower than that, the resulting CI will most likely include 0, suggesting the rare event under study could actually not take place. That is undesired since it omits information which could have been provided with perhaps little more simulation effort. The relative error avoids this problem altogether by ensuring the half-width of the CI will be smaller than $\hat{\gamma}$.

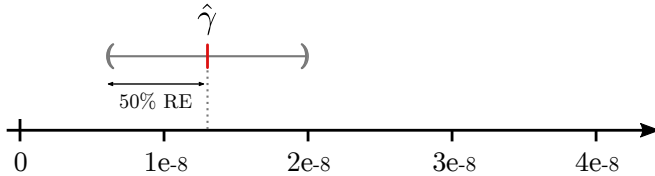


Figure 2.5: Confidence interval built with relative error

Consider now the estimation of a binomial proportion, i.e. γ is the probability of success of some experiment. A simulation will represent an experiment run, and its outcome will be 1 if it succeeded and 0 if it failed, much in line with the examples presented so far. The estimator for γ will be $\hat{\gamma} = \bar{X}$ and the usual estimator for the variance in these cases is $\hat{\sigma}^2 = \hat{\gamma}(1 - \hat{\gamma})/N$. Then for any confidence interval CI

$$\text{RE}_{\text{CI}} = z \frac{\sqrt{\hat{\gamma}(1 - \hat{\gamma})}}{\sqrt{N} \hat{\gamma}} \approx \frac{z}{\sqrt{N} \hat{\gamma}}.$$

The last expression becomes huge for very small values of γ , which is a situation naturally exacerbated by the rarity of the event. Say the user requests a 95% confidence interval and relative error of 10% with $\gamma \approx 10^{-8}$. Then $z \approx 1.96$ and hence N should be greater than 3.84×10^{10} to satisfy the user needs. In this scenario where very few experiments are successful, standard-simulation times can easily become unreasonable. If each simulation takes 1 ms to complete, a computing system with a single execution thread would take 444 days (more than a year!) to satisfy the above criteria.

Modern computers can alleviate this by using parallelism in its various dimensions (ILP, DLP, TLP, etc.) Together with smart implementations and to a certain extent, this can counter the presence of γ in the denominator of Definition 6. Yet there are systems characterised by an *exponential decay*, where polynomial modifications of some model parameter θ (e.g. increase by one the queue capacity) produces an exponential decrease of γ —see e.g. [Gar00,KN99]. In an exponentially decaying regime the standard approach of model analysis by simulation takes an exponential time to converge: for constants $c, k \in \mathbb{R}_{>0}$ one has $T_{\text{std}}(\theta) = O(c^{k\theta})$. An *asymptotically efficient estimator* would instead converge within time polynomial in the rarity parameter: $T_{\text{aef}}(\theta) = O(\theta^{k'})$ for some constant $k' > 0$ [Gar00, Sec. 2.2.1].

The infeasible long times exemplified in the situation above are clearly inherent to the standard Monte Carlo approach, when it is used to analyse a model in a RES scenario. This inefficiency plus the issue of real coverage, i.e. whether or not the theoretical coverage is met by the confidence intervals built, are the main challenges presented by the rare event problem [RT09a,GRT09]. The core of the complication comes from the fact that $0 < \gamma \ll 1$, which implies very few useful paths will be generated during simulation. The two complementary techniques described next have been developed and perfected during the last thirty years to counter this.

Importance sampling modifies the sampling distribution (hence the name) in a way that increases the chance to visit the rare states of the model. This introduces a bias in the resulting point estimate to be corrected with a previously computed *likelihood ratio*. Evidently the change of measure requires a non-trivial understanding of the system under study. Moreover this modification needs to be tractable to allow the computation of the likelihood ratio, meaning it has to be characterised by some function selected ad hoc by the user with certain desired properties like integrability. A bad choice of (change of) measure may have a negative impact on the simulation resulting in longer computation times. In spite of these limiting factors, importance sampling has been successfully applied to several complex and even real-life

systems—see e.g. [GSH⁺92,KN99,XLL07,dVRLR09,LT11].

Importance splitting, also known as multilevel splitting, works by decomposing the state space in multiple layers or *levels*. A level should be higher as the probability of reaching the rare event from its composing states grows, so ideally the rare event would be at the top. Estimation consists in multiplying the estimates of the (not so rare) conditional probabilities of a simulation path moving one level up. The effectiveness of this technique crucially depends on an adequate grouping of states into levels, which is done by some user selected *importance function*. This function assigns a value to each state, its *importance*, which should reflect the likelihood of observing the rare event after visiting that state. So, a state in the rare set should receive the highest importance and the importance of the states decreases according to the probability of reaching the rare event from them.

Most of the critique affecting the change of measure in importance sampling is also applicable to the importance function in importance splitting. This thesis conjectures that building a *good* importance function is *easier* to be carried out by automatic procedures than choosing a *good* change of measure, providing a formalised user query and automata-based model descriptions are available. By “easier” it is meant that fewer assumptions need to be made about the nature of the system, most remarkably there is no need to rely on the memoryless property of the Markovian case. The term “good” is mild in the sense that it only implies an improvement over the standard Monte Carlo approach to analysis by simulation. This thesis does not seek optimality; there are no conjectures about the difficulty of finding an optimal or asymptotically efficient importance function.

Based on that conjecture, we developed algorithms to automatically derive the importance function from the user query and system model description, and present them in the following chapters. To understand the solutions proposed in full detail, a deeper description of the importance splitting technique is presented in the remainder of this chapter.

A last remark is due before concluding the current section. Recall it was earlier stated that importance sampling and importance splitting can be regarded as complementary techniques. On the one hand this is because splitting relies on the possibility to layer the state space, so that the probabilities of crossing “one level at a time” can be computed separately and efficiently. In general this requires long paths between the initial system conditions and the rare event. If the rarity depends on taking very few transitions, each one extremely unlikely to happen, then splitting fails since no efficient layering may be applied to the state space of the model. In such scenario importance

sampling, when applicable, should provide a more natural solution.

On the other hand, when paths from the initial system state need to follow a long and heterogeneous trajectory before reaching a rare state, it can become extremely difficult to choose a change of measure which consistently selects the best transition at each turn. That is why importance splitting can sometimes be the best suited option, particularly in circumstances when many and dissimilar states must be visited before reaching a rare one, or in general when the nature of the model makes it too hard to come up with a tractable and efficient change of measure.

2.5 Importance splitting

In this section several approaches to perform model analysis by simulation employing efficient splitting techniques are described. From here onward the terms *importance splitting*, *splitting technique*, *multilevel splitting*, and the abbreviation I-SPLIT, will be used interchangeably to refer to the approaches for RES described here.

2.5.1 General theory

There are at least two different angles to look at I-SPLIT:

- An original idea by Kahn and Harris was developed from a physical point of view in [KH51], where simulations of particle trajectories were saved and restarted at certain promising states, in order to generate more observations of the rare event. This view bears a strong similarity to another technique introduced by Bayes in the seventies [Bay70, Bay72], which became widespread twenty years later when it was updated and formally extended by José Villén-Altamirano and Manuel Villén-Altamirano, who coined the name *Repetitive Simulation Trials After Reaching Thresholds* (RESTART, [VAVA91, VAMGF94]).
- Another more inherently mathematical overview is to consider the state space of the system as a nested sequence of events, for the formal notion of *event* from probability theory [Bre68, Def. 2.1]. So given $E_0 \supseteq E_1 \supseteq \dots \supseteq E_n$ let the states in E_n define the rare event embedded in the full state space E_0 , and let p_i be the conditional probability that a simulation path reaches E_i given it started from E_{i-1} . Then the

probability of visiting a rare state is the product $\prod_{i=1}^n p_i$ as detailed in [LLGLT09, pp. 42–43].

The notion mentioned first, involving saving and restarting simulation paths, is analysed in depth in Section 2.6 since it is of particular interest for this thesis. The second, more general mathematical definition is described in the remainder of this section, to formally define the splitting technique and introduce some known implementations. All these share the core idea behind splitting to attack the RES problem, but variate to some extent in their approach and properties. Some basic notions on probability theory are required to grasp the following definition. Fundamentals are presented in Appendix B. The reader is referred to [Dur10, Chap. 1] or [Bre68, Chap. 2] for an introduction on the subject.

Formal setting for importance splitting in RES

Suppose the dynamics of the system is described by a stochastic process $X \doteq \{X_t \mid t \geq 0\}$. A *probability space* (Ω, \mathcal{F}, P) and a *measurable space* (S, Σ) are assumed so that each X_t is a random variable on Ω taking values on S , denoted the *state space* or sampling set.

Time t can be either continuous (on the real line) or discrete (on the non-negative integers $\mathbb{N} \doteq \{0, 1, 2, \dots\}$). For convergence purposes in the continuous case, all paths (viz. outcomes of X) are assumed right-continuous with left-hand limits, aka càdlàg.

Moreover and for these definitions, X is assumed to be a Markov process. Since the history of the process can be incorporated inside the system state X_t , this assumption is made without loss of generality for the whole category of time-homogeneous stochastic processes [Gar00, Sec. 2.2].

An *event* will be a measurable subset of the sampling set, i.e. an element of Σ . Let $A \subsetneq S$ be the *rare event* of interest, that is a (measurable) set of states the system can enter with positive but very small probability, e.g. a failure in a digital data storage facility leading to information loss. An event $B \subsetneq S$ is also assumed, denoting some stop or end-of-simulation condition which satisfies $B \cap A = \emptyset$ and $P(B) \doteq P(X_t \in B) > 0$.

Definition 7 (Entrance time). Let $C \subseteq S$ be an event which happens with positive probability, viz. $C \in \Sigma \wedge P(C) > 0$. Then the *entrance time into C* is the r.v. describing the first time that event C is sampled, i.e.

$$T_C \doteq \inf\{t \geq 0 \mid X_t \in C\}.$$

Two ways to analyse systems are of special interest for RES: transient and steady-state analysis. Both are involved with computing or estimating a very small probability value γ , viz. $0 < \gamma \ll 1$, related to the observation of the rare event A . The way of defining γ is what draws the difference between these approaches. This is formalised in Definitions 8 and 9.

Definition 8. In the formal setting described above, the *transient probability of the rare event* will be the probability value

$$\gamma = P(T_A \leq T_B)$$

where T_A, T_B are the entrance times into A, B respectively.

Definition 8 is common in the RES literature—see e.g. [Gar00, Sec. 2.2] and [LLGLT09, Sec. 3.2.1]. It speaks of observing the rare event before reaching some stopping time T_B , here characterised by event B . This can be generalised to any (almost surely finite) time T , which might be of interest when defining simulation truncation not by an event but rather by the passage of time, e.g. “*the probability that a Poke is chosen before T simulation time units elapse.*” Equivalently, time could be included in the state of process X , to speak of a finite time horizon event-wise.

Recall interest lies in the application of I-SPLIT to a formal model description scenario resembling that of model checking. Thus the probability from Definition 8 should be encoded as a user query expressed in some temporal logic. Luckily there is a straightforward mapping from that probability to the PCTL formula

$$P(\neg B \text{ U } A) \tag{7}$$

i.e. “the probability of not observing the stopping condition B until the rare event A takes place.” Notice such query is not pure PCTL since it asks for the numeric probability value rather than whether that value is greater or less than some bound. That is however of no concern since P can appear only at top level and not as sub-expression of neither A nor B , so the last remarks from Section 2.2 apply. Also events A and B need to be encoded as logic formulae. Since PCTL subsumes propositional logic then e.g. A can be simply “ $\neg \text{OVERCROWD}$.” From here onward the *transient analysis* for RES will be an implicit reference to either Definition 8 or eq. (7).

Definition 9. In the formal setting described above, the *steady-state probability of the rare event* will be the probability value

$$\gamma = \lim_{t \rightarrow \infty} P(X_t \in A)$$

also denoted the *long run probability of the rare event*.

Instead of simply writing $P(A)$, the underlying stochastic process X is made explicit in Definition 9 to highlight the dependence on the asymptotic time limit. Steady-state studies have also appeared in the RES literature, most notably in the research around the RESTART splitting technique—see e.g. [VAVA91, VAMGF94]. For a formalisation resembling the one given above the user is referred to [Gar00, Chap. 6], where steady-state analysis is defined in terms of regenerative processes.

Definition 9 asks about the time proportion spent visiting rare states when the system is *in equilibrium*. For Markovian processes it is easy to compute the transition rates that characterise a system in equilibrium, but for general stochastic processes that is usually too hard. The standard simulation approach is to discard some initial system execution path considered transient, and then proceed using the *batch means* technique [LK00], which favours the discard of transient simulation behaviour.

Just like in the transient case, some related formula from a temporal logic is desired to characterise user queries of the probability in Definition 9. The simple CSL formula

$$S(A) \tag{8}$$

talks about “the likelihood of event A in a system in equilibrium,” i.e. the time proportion in the long run that states in A are visited. This fits in CSL in the same way eq. (7) does with PCTL. From here onward the *steady-state analysis* for RES will be referring to either Definition 9 or eq. (8).

In both transient and steady-state analysis for RES, the value γ is positive but very small since the likelihood of reaching its characterizing set A is extremely low. Importance splitting is based on the assumption that there are identifiable *intermediate states subsets* which *must be visited* to reach the rare event, and which are *much more likely* than A to be reached by a simulation path. Formally, the decreasing sequence of events

$$S = E_0 \supset E_1 \supset \dots \supset E_{n-1} \supset E_n = A$$

is assumed, where a projecting function $f: S \rightarrow \mathbb{R}_{\geq 0}$ called the *importance function* determines events E_i . Since it defines the intermediate events which are the cornerstone of the technique, this function is a key component in multilevel splitting. Level values $L_i \in \mathbb{R}_{\geq 0}$ typically called *thresholds* are chosen satisfying $L_i < L_{i+1}$ for $0 \leq i < n$. The events are then defined by means of f and the thresholds:

$$\forall i \in \{0, 1, \dots, n\}. E_i \doteq \{s \in S \mid f(s) \geq L_i\}.$$

Along the thesis and unless noted otherwise it will be $L_0 \doteq 0$ and $L \doteq L_n$, resulting in $A = \{s \in S \mid f(s) \geq L\}$. Furthermore all simulation paths start in the initial state of the system $s_0 \in S = E_0^\dagger$, which should ideally have minimum importance, i.e. $f(s_0) = L_0 = 0$. Nonetheless the more general condition $f(s_0) < L_1$ is sufficient to ensure $s_0 \notin E_1$ as desired.

Notice that in such setting, every simulation path must increasingly traverse all events before reaching a state in A . Besides, considering that $P(E_0) = P(X_t \in S) = 1$ and $E_{i+1} \subset E_i$ for $0 \leq i < n$, the identity

$$\begin{aligned}
 \gamma &= P(E_n) \\
 &= \frac{P(E_n)}{P(E_{n-1})} \frac{P(E_{n-1})}{P(E_{n-2})} \dots \frac{P(E_2)}{P(E_1)} \frac{P(E_1)}{P(E_0)} P(E_0) \\
 &= \frac{P(E_n \cap E_{n-1})}{P(E_{n-1})} \frac{P(E_{n-1} \cap E_{n-2})}{P(E_{n-2})} \dots \frac{P(E_2 \cap E_1)}{P(E_1)} \frac{P(E_1 \cap E_0)}{P(E_0)} P(E_0) \\
 &= P(E_n | E_{n-1}) P(E_{n-1} | E_{n-2}) \dots P(E_1 | E_0) P(E_0) \\
 &= \prod_{i=0}^{n-1} p_i
 \end{aligned} \tag{9}$$

follows by definition of conditional probability, where the *conditional probability of raising one level*, from event E_i into E_{i+1} , is denoted

$$p_i \doteq P(E_{i+1} \mid E_i) \quad \text{for } 0 \leq i < n. \tag{10}$$

The efficiency of multilevel splitting depends on choosing the importance function and the thresholds s.t. $p_i \gg \gamma$ for all i . Thus a stepwise estimation of the p_i can be done more efficiently than an outright estimation of γ .

Since time does not appear explicitly, the previous considerations can be directly mapped to steady-state analysis for a system in equilibrium (cf. [VAMGF94, Sec. 2.2], where the probability of the rare event is defined in a way matching eq. (9) for the renaming $(E_i, p_i, \gamma) \mapsto (C_{i+1}, P_{i+1}, P)$).

Transient studies are based on the same principles. Usually a filtration $\{A_i\}_{i=0}^n$ is defined for $A_i \doteq \{T_i \leq T_B\}$ where T_i is the entrance time into event E_i , so $P(A_n) = P(T_n \leq T_B) = \gamma$ and $P(A_0) = P(T_0 \leq T_B) = 1$. In such setting the conditional probabilities are defined in terms of the filtration: $p_i = P(A_{i+1} \mid A_i)$. This line of analysis reaches an identity analogous to eq. (9). See [Gar00, Sec. 2.4] and [LLGLT09, pp. 42–45] for a detailed study.

[†] An initial probability distribution could also be considered.

Most useful I-SPLIT implementations do not allow a fully independent estimation of the conditional probabilities, because the entrance distribution to E_i affects estimates of p_i to a great extent, conditioning also the estimates for all p_j with $j > i$. Still these probabilities can be computed somewhat separately by taking into account a statistical approximation of the entrance states to each E_i , which resemble the real entrance distributions asymptotically. The general multilevel approach is described next.

With an abuse of notation, event $Z_i \doteq E_i \setminus E_{i+1}$ will henceforth be called the i -th *importance zone* or *level*, and the values $\{L_i\}_{i=0}^n$ will exclusively be referred to as thresholds. So the i -th importance level will be the set of states which the importance function places between thresholds L_i (including it) and L_{i+1} (excluding it). The initial system state will be located in the 0-th (*bottom*) importance level and the rare states will be said to pertain to the n -th (last) level. See Figure 2.6 for a schematic representation.

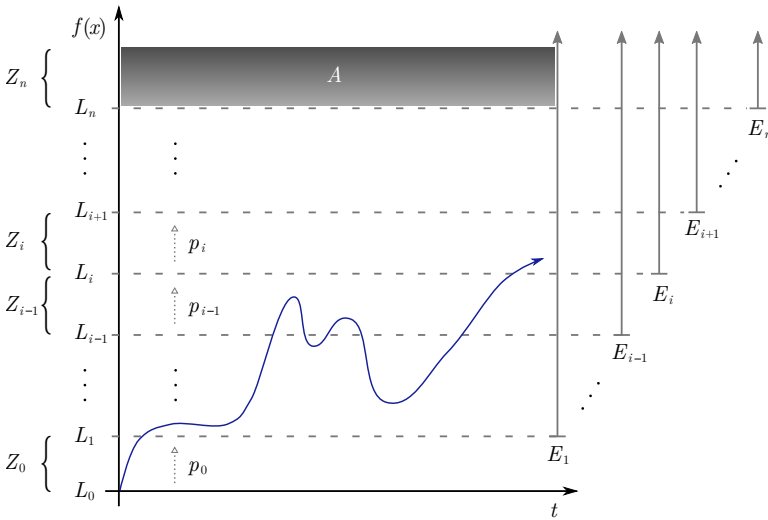


Figure 2.6: Multilevel splitting scenario

Definition 10 (Level-up probability). Let events $\{E_i\}_{i=0}^n$ define the importance levels $\{Z_i\}_{i=0}^n$ in an importance splitting setting. Let S_i be the r.v. with image on E_i which yields the states through which a simulation path can enter E_i . The *probability of moving up from level i into level $i+1$* is the probability that a simulation path visits E_{i+1} , conditioned on the entrance distribution into level i : $P(E_{i+1} | S_i)$.

The level-up probability can be thought of as the probability that a sample path starting at the lower threshold of stage i will hit the upper threshold [Gar00, Sec. 2.4]. Definition 10 helps connecting the mathematical notions defined so far, with the simulation technique which will be described algorithmically. In particular, the following will be of use when proving the unbiasedness of the method.

Proposition 4.

$$P(E_{i+1} \mid S_i) = p_i$$

Proof. The setting in which p_i was defined in eq. (10) involves simulation paths traversing E_i to reach any state in E_{i+1} . Given process X is Markovian, the entrance states into an event E_i fully determine the future of the simulations traversing that region. This means random variable S_i from Definition 10 condenses all information regarding a path traversing E_i . Hence, in the setting of eqs. (9) and (10), conditioning on E_i is equivalent to conditioning on S_i . \square

Basic multilevel splitting approach to RES

Denote by T the time defined by the condition to end the simulation, regardless of whether it is the almost surely finite entrance time T_B from transient analysis, or a finite time horizon (or regenerative cycle duration) in the batch means implementation of steady-state analysis. Denote also by T_i the entrance time into the i -th importance level, viz. the first time a simulation visits a state $s \in S$ s.t. $f(s) \geq L_i$.

Start N_0 independent simulation paths from the initial state of the system model, s_0 . Each of these *original paths* advances until it either reaches T or the entrance time T_1 , whichever happens first. This will be denoted *stage 0* or the *first stage* of multilevel splitting.

Let R_0 be the number of simulations which managed to enter the first importance level, e.g. for which $T_1 < T$. A total of N_0 independent experiments were thus run in this first stage, each having (unconditional) probability p_0 to succeed. Since R_0 counts the number of successes, it has binomial distribution: $R_0 \sim \text{Bin}(N_0, p_0)$. It follows that $\mathbb{E}(R_0) = N_0 p_0$, where $\mathbb{E}(Y)$ denotes the *expected value* of the random variable Y .

Notice $\hat{p}_0 \doteq R_0/N_0$ is an unbiased estimator for p_0 , because $N_0 \in \mathbb{N}$ and thus $\mathbb{E}(\hat{p}_0) = 1/N_0 \mathbb{E}(R_0) = p_0$. Furthermore states $\{s_1^k\}_{k=1}^{R_0} \subseteq E_1$ realizing the *successful trajectories* are an empirical sample of the entrance distribution into E_1 . So each s_1^k is an observation of the r.v. S_1 from Definition 10.

Next, in stage 1, N_1 *simulation replicas* or *offsprings* are started from those R_0 states. In order to maintain a sufficiently large sampling population and assuming R_0 can be small, it is expected that $N_1 > R_0$. By the pigeonhole principle this means more than one simulation may be started from each state. The selection can be done by cloning (or *splitting*) the simulations that reached each s_1^k , or choosing randomly where to start each of the N_1 simulations from the R_0 available options.

Each new trajectory is again simulated from its starting state until either T or T_2 occur, whichever happens first. Let R_1 be the number of simulations where $T_2 < T$. Stage 1 thus consists of N_1 experiments, and the r.v. R_1 counts the successful simulations which reached the second importance level.

Nonetheless, a binomial distribution cannot be unconditionally assumed, because not all simulations are necessarily independent. Some may have started from the same state s_1^k , sharing their history up to that point.

Recall however that states $\{s_1^k\}_{k=1}^{R_0}$ have an asymptotic behaviour described by the distribution of S_1 . Thus when conditioned on such random variable, stage 1 can indeed be regarded as a Binomial experiment.

To gain on intuition, think that knowing the full history back to s_0 , where the original N_0 *independent* simulations were bootstrapped, suffices to grant the statistical independence sought in the simulations of stage 1. Moreover, conditioning on S_1 is reasonable because the starting states of stage 1 are an empirical sample of that random variable.

By Proposition 4, each of the N_1 launched simulations succeeds with probability $P(E_2 | S_1) = p_1$. Furthermore $\mathbb{E}(R_1 | S_1) = N_1 p_1$. Consider the estimator $\hat{p}_1 \doteq R_1/N_1$, then clearly $\mathbb{E}(\hat{p}_1 | S_1) = 1/N_1 \mathbb{E}(R_1 | S_1) = p_1$.

Generalizing this approach, at the i -th stage N_i simulation paths are launched from the R_{i-1} previously successful trajectories. This is repeated until the last level is reached. By then all estimates $\{\hat{p}_i\}_{i=0}^{n-1}$ will have been computed, whose product provides an estimator for γ :

$$\hat{\gamma} = \prod_{i=0}^{n-1} \hat{p}_i . \quad (11)$$

Interestingly, even though eq. (11) yields an unbiased estimate of γ as it will be shown next, the intermediate proportions $\hat{p}_i = R_i/N_i$ depend on each other as indicated (\hat{p}_0 aside). This is a consequence of the dependence of i -th stage paths on the entrance distribution to the i -th importance level.

This dependence can be very strong if the importance function and the thresholds are not chosen carefully, which can reduce greatly the efficiency of the splitting technique. More on this in Section 2.7.

Unbiasedness of the estimator

The core idea is to exchange product for expectation in the sequence $\{\hat{p}_i\}_{i=0}^{n-1}$ of estimates from eq. (11). Notice first that the previous remarks regarding the unbiasedness of \hat{p}_1 can be extrapolated to any estimate, as long as the full history of trajectories up to its corresponding importance level is known.

For $1 \leq k < n$ denote \mathcal{F}_k the σ -algebra associated with the stochastic process $\{S_i\}_{i=1}^k$, then

$$\mathbb{E}(\hat{p}_i \mid \mathcal{F}_i) = p_i$$

for all $0 < i < n$ (cf. [Gar00, equations 2.5 to 2.8]). Consequently by the law of total expectation, for any two different indices $i, j \in \{0, \dots, n-1\}$ (assume $i < j$ w.l.o.g., cf. [Gar00, eq. 2.13])

$$\begin{aligned} \mathbb{E}(\hat{p}_i \hat{p}_j) &= \mathbb{E}(\mathbb{E}(\hat{p}_i \hat{p}_j \mid \mathcal{F}_j)) \\ &= \mathbb{E}(\hat{p}_i \mathbb{E}(\hat{p}_j \mid \mathcal{F}_j)) \\ &= \mathbb{E}(\hat{p}_i p_j) \\ &= \mathbb{E}(\hat{p}_i) p_j \\ &= \mathbb{E}(\mathbb{E}(\hat{p}_i \mid \mathcal{F}_i)) p_j \\ &= p_i p_j. \end{aligned}$$

Theorem 5 (Unbiasedness of I-SPLIT, [Gar00]). *In the approach described above, the expected value of the estimator from eq. (11) is the probability of the rare event from eq. (9), viz.*

$$\mathbb{E}(\hat{\gamma}) = \gamma.$$

Proof. Recursively applying the previous argument one gets $\mathbb{E}(\prod_{i=0}^{n-1} \hat{p}_i) = \prod_{i=0}^{n-1} p_i$. The desired equality follows by equations (11) and (9). \square

2.5.2 Variants of the basic splitting technique

There are many ways to implement multilevel splitting. The basic approach described in Section 2.5.1 is just one example well suited for introductory purposes and for proving unbiasedness. A neatly organised overview of various implementation alternatives is presented in [LLGLT09, Sec. 3.2.2], of which a reviewed summary is shown next.

How to choose the number of offsprings N_i at each stage is a pivotal decision. Typical strategies are:

- *Fixed splitting.* In the i -th stage, each successful simulation path reaching the upper threshold generates the same number of offsprings $K_i \in \mathbb{N}$. The total number of simulation paths $N_{i+1} = R_i K_i$ in the next stage is thus a random variable. This is sensitive to both the $\{K_i\}_{i=1}^{n-1}$ and the $\{R_i\}_{i=0}^{n-1}$. If $R_i = 0$ the technique suffers from *starvation* since no offsprings will be produced. If $K_i \gg N_{i+1}/R_i$ then too many offsprings will be produced and the technique suffers from computational *overhead*.
- *Fixed effort.* A predetermined number $N_i \in \mathbb{N}$ of offsprings is started during the i -th stage. If $R_{i-1} < N_i$ then these offsprings can be assigned to the available R_{i-1} starting states randomly or deterministically. This rules out the possibility of overhead, but can suffer from starvation if N_i is too small to ensure $R_i > 0$.
- *Fixed success.* The number $R_i > 0$ of successful simulation paths in the i -th stage is predetermined. Thus N_i is a random variable for the i -th stage, since sufficient simulations need to be launched to reach the desired R_i . This can cause computational overhead but cannot suffer from starvation by definition.

These three strategies have different performance implications. Fixed splitting can be considered lightweight w.r.t. memory consumption, since it allows a *depth-first search* (DFS) implementation [LLGLT09, p. 45]. Namely, during the first stage each original path is simulated until $\min(T, T_1)$. If T_1 takes place it yields K_1 offsprings from that path; then each of these offsprings is simulated until $\min(T, T_2)$, and so on, before moving on to attend the next original path.

Such DFS approach cannot be applied to the other two alternatives, which need to attend one importance level at a time and keep in memory all resulting entrance states into each level.

In the basic approach introduced, all simulation paths have the same weight at any importance level. In a fixed splitting scenario, consider a more general setting where trajectories can be assigned different weights. Each of the N_0 original trajectories in the bottom importance level will have weight 1. During the next stage in the first importance level, each offspring will have *relative weight* $1/K_1$, since it comes from an original trajectory with weight 1 that was split K_1 times.

This means the successful paths in the uppermost level will have relative weight $(\prod_{i=1}^{n-1} K_i)^{-1}$. Then the estimator $\hat{\gamma}$ is the sum of relative weights of these final successful trajectories divided by N_0 .

Such generalised approach, which takes the relative weights of the simulation paths into account, is of special use when the rare event can appear in low importance levels. More on this in Section 2.6.

Estimator $\hat{\gamma}$ from eq. (11) is *efficient*, because its variance is smaller than the standard Monte Carlo estimator of γ [Gar00, Sec. 2.4.3]. A smaller variance means less samples are needed to converge. Nevertheless, when practical applications are considered, the computation time of each sample has a direct impact on the convergence wall-clock time.

For instance in fixed effort and fixed success, paths are simulated until the upper threshold or final time T are met. In transient analysis the average computational time to reach T may increase significantly with the importance level i where the path started [LLGLT09, p. 46].

Symmetrically, the maximum computational parallelism can act as bottleneck in fixed splitting. Since new paths are injected each time an upper threshold is reached, there is a risk of having an exponential explosion in the resulting number of concurrent trajectories.

To keep at bay the computational overhead derived from potentially long simulation paths, early path termination (aka *path truncation*) is a typical strategy. The essence of path truncation is to select and kill ideally unpromising trajectories, before its “natural cause of death” (e.g. reaching T) takes place. Several strategies have been studied:

- *Deterministic truncation.* For some selected *die-out depth* $\beta \in \mathbb{N}$, kill any trajectory that submerges more than β levels from its creation level. That is, if some path originated from an entrance state into E_i , it will be truncated as soon as it visits a state in level $i - 1 - \beta$ or lower. This requires a proper weighing of paths to avoid introducing a bias in the estimation.
- *Probabilistic truncation.* For die-out depth β , i -th level paths go through a *Russian roulette* test each time they down-cross a level deeper than $i - \beta$. More precisely, numbers $\{r_{i,j}\}_{j=\beta}^{n-1} \in \mathbb{R}_{>0}$ are chosen for paths originated in the i -th importance level. These are truncated with probability $1 - 1/r_{i,j}$ as they cross level $i - 1 - j$ downwards for $j \geq \beta$. On survival their weight is multiplied by $r_{i,j}$. To reduce the variance introduced by weighing, a simulated trajectory of weight w is cloned $w - 1$ times as it reaches the uppermost level[†]; then each of these newly independent paths will have weight 1.

[†] Non-integral w require special treatment, see [LLGLT09].

- *Periodic truncation.* Similarly to the probabilistic version, numbers $\{r_{i,j}\}_{j=\beta}^{n-1} \in \mathbb{R}_{>0}$ are chosen for i -th level trajectories and global $\beta \in \mathbb{N}$. To reduce the variability of the Russian roulette approach numbers $D_{i,j}$ are uniformly chosen once among $\{1, \dots, r_{i,j}\}$. Then for i -th level trajectories, every $(r_{i,j} D_{i,j})$ -th path to go down level $i - 1 - j$ for $j \geq \beta$ is retained and its weight is multiplied by $r_{i,j}$; all other i -th level paths to do so are killed.
- *Tagged truncation.* Each i -th level path is *tagged* to the importance level number $(i - 1 - j)$ with some probability which increases with j for $j \geq \beta$. Trajectories are truncated iff they visit their tagged levels.

Popular implementations

Some selections of the above criteria have been thoroughly studied and successfully applied to several case studies, becoming somewhat conventional in the RES community. Three versions of such implementations of importance splitting are briefly described next.

- *RESTART* is a method developed by the Villén-Altamirano brothers and covered in depth in the next section. It follows a DFS approach which uses fixed splitting when a simulation path reaches a threshold upwards. As this happens a single offspring is tagged as the *original path* which came from the previous level. Truncation is deterministic with $\beta = 0$, i.e. any copy of the original path that crosses downwards its creation level is killed. This reinforces the idea of favouring promising runs and discarding unpromising ones. To avoid starvation the original path from the previous level is spared as it goes down, somehow resembling tagged truncation. There is a single original simulation path for each RESTART run with weight 1, and the relative weighing scheme explained above is used for $N_0 = 1$.
- *Fixed Effort* is a term coined by Marnix Garvels in his Ph.D. thesis to refer to a *breadth-first search* (BFS) approach to I-SPLIT much in line with the basic setting initially described in Section 2.5.1. It has also been called plainly *splitting* in a comparison against RESTART [VAVA06]. It consists of an incremental approach starting at the bottom importance level, which truncates simulations as soon as they reach the stopping time T or the upper importance level. The entrance states into the next level, pinpointed by the successful paths which were truncated

upon reaching the upper threshold, are saved and used as starting points for the simulations in the the next stage. Each importance level is covered in this way, one at a time, until the rare event is reached. By then, estimates \hat{p}_i for the conditional probabilities of Definition 10 have been computed for each level, which are multiplied to estimate the rare event following eq. (11).

This method uses fixed effort with deterministic choice as offspring generation mechanism, plus deterministic path truncation. Notice in the standard implementation paths are not truncated when they go down an importance level, as it is done in RESTART, unless this happens together with T . Moreover no weighing is necessary since trajectories are not allowed to cross thresholds; in that respect all they do is determine the starting states for the next level simulations when they successfully reach the upper threshold.

- *Adaptive Multilevel Splitting* [CG07] and its successor *Sequential Monte Carlo* [CDMFG12] are harder to place in the current picture, since they skip the pre-selection of thresholds $\{L_i\}_{i=1}^n$ by discovering them dynamically while simulation paths are pushed towards the rare event. Here the user is asked the desired level up probability p_i a priori, making the number of levels n the random variable to estimate.

For the sake of clarity let $p = p_i$ for all levels $i \in \{0, \dots, n\}$, where the total number of levels n is unknown. Initially starting from stage 0, at the i -th stage m independent simulation paths start from m predefined states (e.g. in stage 0 these m states are the initial state), and run until they meet some termination criteria, e.g. reaching T . Each path visited several states with different importance. Let v_i^j be the highest importance value seen by the j -th path, then the i -th stage yields the data set $V_i = \{v_i^j\}_{j=1}^m$. For $k = \lceil pm \rceil$, let ν_i^k be the $(m - k)$ -th m -quantile of V_i . That is, for V_i seen as an array sorted in increasing order, let ν_i^k equal the value in the $(m - k)$ -th position. Then ν_i^k is the candidate for *upper threshold* of the i -th stage, because a simulation will reach it with probability p (roughly). Furthermore, any state which has importance ν_i^k and was visited during these runs, is a potential starting point for the simulations in the next stage.

Eventually the rare event is reached and the number of stages n determined, yielding the rare event estimate $\hat{\gamma} \doteq p^n$. These implementations are ideal for continuous state spaces, with potential practical problems when applied to discrete models.

2.6 RESTART

In the literature about RES, one of the best known versions of importance splitting is the RESTART method. Already in 1970 A. J. Bayes introduced an accelerated simulation method to estimate the probability of a stochastic process being in a state of a rare set [Bay70], with many of the properties mentioned above. Twenty years later in 1991 José and Manuel Villén-Altamirano rediscovered the method in [VAVA91] coining the famous acronym. Later on they first generalised it to work with multiple thresholds in [VAMGF94], and then in [VA98] to handle a rare event which can occur inside any importance level. Both the versatility of the technique and its relative ease of implementation make it a perfect candidate for the general approach sought in this thesis. A deeper insight of its characteristics is hence presented below.

A thorough explanation of (a mature version of) RESTART can be found in [VAVA11, Sec. 2], a transcription of which is given next using a notation more compliant with the one we have presented so far. A Markov process $X = \{X_t \mid t \geq 0\}$ is assumed, and thresholds $\{L_i\}_{i \geq 0}^n$ are defined on the real line, determining *importance regions* $S = E_0 \supset E_1 \supset \dots \supset E_n \supseteq A$ for a rare set A . This is done via an importance function $f: S \rightarrow \mathbb{R}$ which maps the state space S of X into the real line, so $E_i = \{s \in S \mid f(s) \geq L_i\}$. Notice “region E_i ” here is the same as “event E_i ” in the formal setting previously introduced for general I-SPLIT. Likewise, importance zones $Z_i \doteq E_i \setminus E_{i+1}$ (denoting $Z_n \doteq E_n$) create a partition of S where the higher the value of i , the higher the importance of the states contained.

With an abuse of notation and exclusively when talking about the execution of RESTART, we will use the term *event* to refer to a *simulation incident*, i.e. an occurrence that changes the state of the underlying Markov process. So, given a simulation path traversing S , let a *rare event* or *A event* refer to this path taking a transition whose target is a rare state $s \in A$. Define an *E_i event* in the same way for any region E_i . Let also a *B_i event* denote the path taking a transition $s \rightarrow s'$ whose originating and target states satisfy $f(s) < L_i$ and $f(s') \geq L_i$ respectively. That is, a *B_i event* tells when the simulation has “gone up” into the i -th importance region. Equivalently define a *D_i event* when the simulation “goes down” from the i -th importance region into any zone Z_j with $j < i$. RESTART works as follows:

1. A simulation path called the *main trial* starts from the initial system state $s_0 \in Z_0$. This path will last until a predefined end-of-simulation condition is met, say a finite time horizon T or an almost surely finite entrance time into a stopping set.

2. Each time an event B_1 occurs in the main trial the system state is saved, the main trial is interrupted, and $K_1 - 1$ offsprings or *retrials of level 1* are generated.
 - ▷ A retrial is just an independent simulation path which originates from the entrance state into some higher region by another trial. In this case, by the main trial entering E_1 .
 - ▷ A retrial of level 1 is truncated when it causes a D_1 event (viz. goes down to zone Z_0) or meets the end-of-simulation condition.
 - ▷ Notice the execution thread of the computer switched from the main trial to its offsprings, following the DFS approach described for fixed splitting in Section 2.5.1. An equivalent mechanism will be set in motion as these offsprings generate B_2 events.
3. After the $K_1 - 1$ retrials have been attended until truncation, the main trial is restored at the saved state from the B_1 event.
 - ▷ Including this original trial, the total number of simulated paths between events B_1 and D_1 is K_1 . Each of these K_1 trajectories is called a $[B_1, D_1]$ -trial.
 - ▷ Only the main trial can continue after D_1 , potentially generating new B_1 events and thus avoiding starvation.
4. Each $[B_1, D_1]$ -trial could have triggered a B_2 event during its execution. As this happens an analogous process is set in motion: $K_2 - 1$ offsprings of level 2 are launched, starting in the state which caused B_2 and finishing in a D_2 event.
 - ▷ The trial from level 1 that generated the B_2 event is *the original trial of level 1*, and is the only one that will survive a D_2 event.
 - ▷ Just like the main trial before, the original trial of level 1 can then generate more B_2 events. It will be killed however if it generates a D_1 event, since it is a retrial of level 1 and thus a $[B_1, D_1]$ -trial.
 - ▷ Counting the original trial of level 1, there are K_2 trials of level 2, denoted $[B_2, D_2]$ -trials.
5. In general for $1 \leq i \leq n$, $K_i \in \mathbb{N}$ is the number of $[B_i, D_i]$ -trials launched each time a B_i event is triggered by a $[B_{i-1}, D_{i-1}]$ -trial.
 - ▷ K_i is called the *i -th splitting factor* or *splitting value*. It is a constant chosen a priori by the user, with the restriction $K_i > 1$.

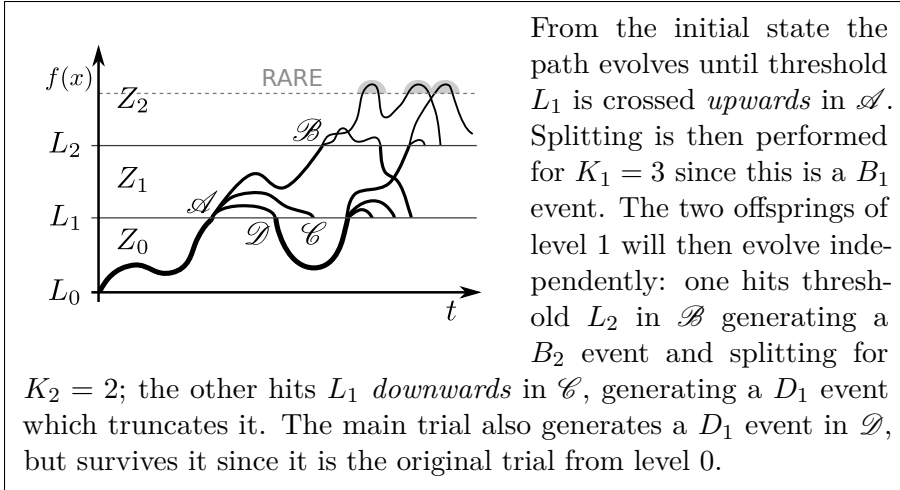


Figure 2.7: Schematic representation of a RESTART run

The method as described above relies on an ideal implementation, where the rare event is entirely contained in the uppermost region and a simulation path can rise by at most one importance region at each step. In such setting any trajectory visiting a rare state has traversed all splitting stages, which stacked up on every threshold crossed. An unbiased estimator is obtained applying the relative weighing scheme with a weight equal to 1 for the main trial. Thus the relative weight of a level n retrial producing a rare event in zone Z_n is $1/K$, for the *stacked splitting factor* $K \doteq \prod_{i=1}^n K_i$.

Notice that, if simulations were monotonically increasing in importance, the stacked splitting factor is actually the maximum number of offsprings which could be concurrently running in the uppermost importance region. Thus K can also be introduced as the *statistical oversampling* incurred by the offsprings of level n that visit the rare set A .

To provide an explicit formula for an estimator derived from a RESTART simulation, consider first a steady-state analysis in a continuous time model. Say M retrials of level n eventually make it to the rare set. For some finite time horizon $T < \infty$ of a batch means run, say the (simulation) time each retrial spent on the rare event is $\{t_j^*\}_{j=1}^M$. Then given $T^* \doteq \sum_{j=1}^M t_j^*$, an unbiased estimator for the time proportion (viz. the steady-state probability) of the rare event is

$$\hat{\gamma} \doteq \frac{T^*}{KT}$$

corresponding to a single batch means execution of T time units of simulation, where K is the stacked splitting factor. Proofs for the unbiasedness of this estimator can be found in [VAVA02, VAVA11][†].

RESTART can also be applied to transient analysis, obtaining an equally unbiased estimator—see e.g. [GHSZ98, GHSZ99, GK98, GVOK02]. The idea is to launch N_0 main trials instead of the single one of steady-state analysis, since each trial is expected to be short lived. Say M retrials of level n make it to the rare set A before entering the stopping set B . These have benefited from the stacking up of the splitting mechanism, thus each has relative weight $1/K$. There is no permanence time to measure, since simulations are truncated as soon as they visit a rare state; what counts is them having reached A before B . So M can be thought of as the number of successes in a pseudo binomial experiment where each single experiment is of RESTART nature rather than Bernoulli. The estimator in this setting is:

$$\hat{\gamma} \doteq \frac{M}{K N_0}$$

where K accounts for the statistical oversampling, acting as relative weight of the M successful simulations.

Care must be taken when studying transient properties with RESTART under this pseudo binomial perspective. Compared to the binary outcome of a standard Bernoulli experiment, a single RESTART run has a potentially unbounded outcome. Think e.g. of a situation where the main trial goes up and down the first threshold repetitively: then arbitrarily many B_1 events could be generated, whose spawned offsprings may visit the rare event in enough proportion to ensure $M > K$. Furthermore, not only could the outcome of a RESTART run be greater than 1, but it could also take any value in $\{0, 1/K, 2/K, \dots, K-1/K, 1\}$, due to the weighing of K .

For these reasons, performing transient analysis with RESTART cannot be strictly regarded as estimating the proportion p of a Binomial experiment. As a consequence, when computing confidence intervals around the point estimates generated by the application of RESTART, the usual strategies for Binomial proportions (e.g. using the Wilson score interval or the Agresti-Coull interval) cannot be directly applied.

In spite of such complications, one must remember that our interest lies in the expected behaviour of the technique: the estimators given above

[†] An interesting generalised proof is given in *Recent Advances in RESTART Simulation*, a seminar the Villén-Altamirano brothers presented in RESIM 2008.

are unbiased because their expectations converge to the desired population parameters. For steady-state analysis this means that prolonging the total simulation time T will draw the estimate closer to the true long run behaviour of the system. For transient analysis it is the number of main trials N_0 that must be increased, in order to obtain a more significative estimate.

As mentioned before, all of the above applies to an ideal implementation of the technique, where it is assumed a simulation path can rise by at most one importance region at each step. Generally speaking the definition of the Markov process X and the choice of importance function may allow a simulation path to jump over some importance zone, viz. taking a transition $s \rightarrow s'$ with $s \in Z_{i-1}$ and $s' \in E_{i+1}$. In such cases it must be considered that several B_i events occurred simultaneously, and the corresponding splitting, tagging, and saving of states must be dealt with accordingly.

For instance, say transition $s \rightarrow s'$ jumps over zone Z_i , e.g. $f(s) < L_i$ and $f(s') = L_{i+1}$. Since that is a B_i event, $K_i - 1$ retrials of level i have to be launched starting in state s' and finishing when an event D_i occurs. Yet taking that transition also means each of those trials (including the original one from level $i - 1$) is causing a B_{i+1} event. Since the total number of $[B_i, D_i]$ -trials is K_i , then $K_i(K_{i+1} - 1)$ retrials of level $i + 1$ are also started from state s' , which will finish when an event D_{i+1} takes place. In total there are thus $K_i K_{i+1}$ simulation paths: the original trial from level $i - 1$; the $K_i - 1$ retrials of level i which will be killed by a D_i event; and the $K_i(K_{i+1} - 1)$ retrials of level $i + 1$ which will be killed by a D_{i+1} event.

Another potential yet realistic complication is having a rare event which can occur in any importance region, not only E_n [VA98]. From the point of view of the implementation this can be countered quite easily. It suffices to consider the relative weight of the simulation paths visiting the rare states, which here need not be $1/K$ but will be $W_\ell \doteq 1/\prod_{i=1}^\ell K_i$ for a rare state in zone Z_ℓ with $\ell \leq n$. A proper tagging of the simulations allows to inject this update into the estimators above described: retrials of level i will be tagged with weight W_i and the global division by K is replaced by multiplication with the corresponding W_i , as in [VA98, eq. (2)].

Notice however that in a scenario where a rare state lies in zone Z_ℓ for $\ell < n$, the sampling of the rare event will not benefit from the full power of the splitting procedure. The efficiency of the method hence deteriorates, as analysed in depth in [VA98, VAVA11].

2.7 Applicability and performance of I-SPLIT

RESTART is indeed versatile as to the scope of system models where it can be applied—see e.g. [VAMGF94, GK98, GHSZ99, VAVA06, VA07b, VA07a, VA09, VAVA13, VA14, BDH15]. It is nonetheless relevant, as with any other splitting technique, to know how efficiently it can be applied in each case.

In [GHSZ98, GHSZ99], sufficient conditions for an asymptotically efficient application of RESTART are provided, one of whose basic hypothesis is to work only with countable-state Markov chains. These kind of restrictions, commonplace in the literature due to the good properties of the memoryless distribution, are too strong for the general objectives of this thesis.

The next chapters present automatable techniques for the implementation of multilevel splitting, which aim at covering the broad scope of (time-homogeneous but otherwise general) stochastic processes. This automation should nonetheless introduce as few restrictions as possible. Recall the goal is not optimality but rather applicability of the splitting technique, in any way that outperforms the standard Monte Carlo approach that was detailed in Section 2.3.2.

The variance of the estimators is the most usual theoretic instrument employed to measure and contrast the performance of different estimation mechanisms. Such variance has a direct impact in the precision of the confidence intervals produced, and thus (generally) in the convergence times. In consequence, to decide whether RESTART is a good candidate for experimentation with I-SPLIT, some mathematical characterization for the variance of its estimators is desirable.

The authors Manuel and José Villén-Altamirano have developed several expressions for such variance, some of them reported in [VAVA02]. They all are purely theoretical in nature, meaning they cannot be effectively computed in real-life applications, at least not for general stochastic models. Nevertheless they show that for optimal, quasi-optimal, and even merely good implementations of the method, there is indeed an expected gain in using RESTART over standard Monte Carlo simulation [VAVA11].

Remarkably, the optimal and quasi-optimal selection of parameters proposed in [VAVA11] are impractical due to the assumptions these make on the systems [VAVA11, Sec. 5]. Notwithstanding this inconvenience, a good implementation of RESTART can anyhow be performed. Specific guidelines for such an “effective application” of the method are given in [VAVA11], fitting the objectives of this thesis wonderfully. Specifically, four main distinct factors affecting performance are reported:

f_O Inefficiency due to the computing overhead.

This is related to the concrete implementation of the computational methods, which depends also on the model. It is affected by the evaluation of the importance function every time a state is visited (e.g. at each simulation step), the comparison of the resulting importance to the threshold values, and the context switch for saving and restoring states during replication and truncation. So for instance the number of variables of the system influences RESTART negatively. There is no universal solution for this source of inefficiency: in general smaller models should be favoured, and good design patterns and programming techniques are paramount to minimise the overhead derived from state manipulation and from the evaluation of the importance function.

f_K Inefficiency due to the splitting values.[‡]

When discussing the fixed splitting strategy for offspring generation in Section 2.5.1, it was said that a careless selection of the splitting factors $\{K_i\}_{i=1}^n$ can lead to overhead or starvation. Optimal and quasi-optimal values for all K_i require a dense state space S . For the general case there is a procedure based on pilot runs starting from the initial state s_0 , which increasingly chooses the K_i trying to fulfill the *balanced growth* equation [Gar00, eq. (2.25)]. This requires a previous fixing of the thresholds and operates with a balancing procedure: when the i -th threshold is granted a splitting value bigger than desired (e.g. due to the discreteness of the state space), it will be compensated with a smaller K_{i+1} , and vice versa. The performance incidence of an error in this selection mechanism should be only moderate [VAVA11].

f_L Inefficiency due to the threshold levels.[§]

The selection of the number and location of the thresholds is similar in impact to the choice of splitting values, since these two parameters are intimately related in a fixed splitting strategy. Up to a certain point, choosing thresholds too close to each other can be countered by reducing the splitting. Nevertheless, setting them too far apart may incur in unavoidable starvation: notice the extreme case of a single threshold at the boundary of A , which would turn I-SPLIT into standard Monte Carlo simulation. In this respect, the authors of RESTART recommend

[‡] In [VAVA11] this appears as factor f_R ; here it is renamed to fit the current notation.

[§] In [VAVA11] this appears as factor f_T ; here it is renamed to fit the current notation.

using pilot runs and statistical analysis, trying to fix the values $\{L_i\}_{i=1}^n$ so that the probability of level up is near $1/2$. If the nature of the model makes this impossible, e.g. the probability of a single transition $s \rightarrow s'$ is already lower than 0.5, the thresholds are set as close as possible to each other. A subsequent choice of splitting values will try to counter such situations using the balanced growth heuristic.

f_V *Inefficiency due to the variance of the B_i events.*

This factor speaks of the variance in the *true importance* of a B_i event, as the event is triggered by different entrance states into region E_i . That is, the unknown theoretical probability of observing a rare event after visiting the state that caused the B_i event. If this importance varies much with the entrance states into the different regions, the performance of the technique can deteriorate greatly, since the splitting at the i -th threshold will not cause an homogeneous oversampling, and the outcomes of independent RESTART runs could variate significantly. Even worse, some trajectories may be favoured over others, which could yield an incorrect estimation. This in spite of the unbiasedness of the method, since the computation of the CI could prematurely converge to a wrong estimate, because no trajectories have yet been sampled from an unlikely but representative set. RESTART is quite sensitive to this factor, which is affected by the concrete modelling of the system and the choice of the importance function $f: S \rightarrow \mathbb{R}$.

From the four factors exposed, the last one, f_V , is the most difficult to counter systematically. Guidelines are provided in [VAVA11] to reduce it, but seem difficult to generalise outside the scope of that work. That is because it depends on the nature of the particular problem under study, which has mostly led to ad hoc solutions, very well suited for the situations where they are proposed, but inapplicable in a different scenario. As stated in Section 2.1, formal system modelling offers several tools to structure the description of a system. This could alleviate the f_V problem, inasmuch a highly-structured description of the model can be produced, with little variability among the real importance of the states that lead to a B_i event.

Yet even in the best scenario, there remains the issue of the choice of an importance function. This too has mostly been dealt with in a per case basis: most articles on importance splitting propose ad hoc functions for some selected case studies and show how well (or bad) these perform. Computation of generic functions is still mostly a novel field, with the exceptions mentioned in Section 1.2.

Evidently, the performance implications of a good choice of importance function go beyond RESTART; factor f_V above is just a concrete example of how critical this can be. Since all splitting techniques guide simulations in an attempt to visit the states with the highest (computed) importance, *a bad function choice will result in a bad splitting technique*, regardless of its particular characteristics. This will be theoretically and empirically illustrated in the following chapters.

As earlier stated, the general motivations of this thesis involve automating the analysis by simulation of general stochastic models under rare event conditions. Given the great potential for innovation and the direct impact it could have in industry, *automatic importance function derivation* for the application of importance splitting techniques to RES is a promising area of research, and the main specific goal of this thesis.

Automatic I-SPLIT: monolithic approach

3

This chapter presents two main contributions of the thesis: how to derive an importance function from a global model of the system, and how to use that function in an automatic approach for multilevel splitting. The relevance and sensitivity of the choice of importance function for I-SPLIT is illustrated by means of some practical examples. Then, a formal framework for system modelling is determined, upon which the derivation algorithm is defined. An automated implementation of the full analysis process for RES is introduced after that. Finally the efficiency of this technique, as well as its major limitation, are demonstrated by means of case studies.

3.1 The importance of the importance function

The same formal setting from Section 2.5 will be used here. That is, there is some time-homogeneous stochastic process $X = \{X_t \mid t \geq 0\}$, with discrete or continuous time t , describing the model under study. For the following definitions X is required to satisfy the Markov property. This can be done w.l.o.g. since the history of the system can be included into the state X_t of the process—see [Gar00, Sec. 2.2].

The state space is denoted S and the rare set is $A \subset S$, which the process samples with positive but very small probability $0 < \gamma \ll 1$. *Events* are measurable subsets of S , and A is also denoted the *rare event*. The general goal in rare event simulation is to compute the probability value γ of observing the rare event, using statistical analysis on some set of paths simulated from the initial system state $s_0 \in S \setminus A$. Both transient and steady-state analysis are of interest as introduced by Definitions 8 and 9.

The splitting technique relies on a decomposition of S grouping together states with similar probability of leading to the rare event. The computational approach is the simulation of paths, so this speaks of the probability that a path visiting a state will eventually reach the rare event. Furthermore since X is Markovian, it is enough to just consider paths *starting* from each state.

From this point of view a visit to A can be seen as a goal to achieve, and the *grade of importance* of a state $s \in S$ would reflect the likelihood of achieving the goal when paths start from s . The following definition formalises a quantification of this property.

Definition 11. For time-homogeneous Markov process X , let $A \subset S$ be a rare event in its state space. The *true importance* of a state $s \in S$ is the probability that a simulation path starting from s will visit some state in A , viz. that a r.v. from X observes event A , conditioned on the initial state s .

Notice the true importance of the initial state s_0 is precisely γ . Evidently these values are unknown in general but for the states in A , and one of the goals of RES is to estimate them. In importance splitting this is done for s_0 employing some predefined scaled approximation of such values, which must cover the full state space. Such (arbitrary) approximation is known in the literature under the name of *importance function* (also score function [JLS13] and rate function [GHSZ98]), and it can be any projection which maps the states into some totally ordered set or field:

Definition 12. An *importance function* (I-FUN) is a mapping

$$f: S \rightarrow \mathbb{R}.$$

For each $s \in S$ the value $f(s) \in \mathbb{R}$ is the *computed importance* or simply the *importance* of s .

The quality of this approximation is strongly correlated to the performance of the technique. The more it resembles the (scaled) true importance of the states, the faster the method should converge—see Section 2.7. As a matter of fact once the system model has been formalised, deciding on an importance function is usually the first step for the application of multilevel splitting. Most techniques even have procedures to select and tune other execution parameters based on a user-provided definition of the I-FUN. Take for instance [CG07] which only needs the choice of ratio k/n besides this function, or the guidelines given in [VAVA11] to derive the thresholds and the splitting factors for a practical application of RESTART.

Unfortunately Definition 12 leaves all the work to the implementer. Any heuristic should assign the importance values coherently with Definition 11, but this is infeasible from a practical viewpoint, otherwise the solution to the problem would be known already. Historically, the most popular way to make this choice is defining the function ad hoc given the particular system under

study. Be that as it may and on top of its inextensibility, such approach requires a qualified human decision which could nonetheless be mistaken, with significant performance implications. The sensitivity of I-SPLIT to the choice of importance function is illustrated in the following practical setting.

Example 1: Queueing system with breakdowns.

Consider the Markovian queueing system from Figure 3.1 where there is a single buffer, `buf`, to which several sources send packets concurrently. The sources can be of type $i \in \{1, 2\}$ and have exponentially distributed on/off times characterised by parameters α_i and β_i respectively. When active, sources of type i send packets to the system bus at rate λ_i , which are immediately enqueued in `buf`. Enqueued packets are handled by a server at rate μ as long as it is operational. The server breaks down periodically at rate ξ and gets repaired at rate δ .

This system was originally studied in [KN99] for an initial state with a single enqueued packet, a broken server, and all sources down but for one of type 2. Using importance sampling the authors estimated the transient probability of reaching a parametric maximum capacity $K \in \mathbb{N}$ in the buffer before the server could process all enqueued packets.

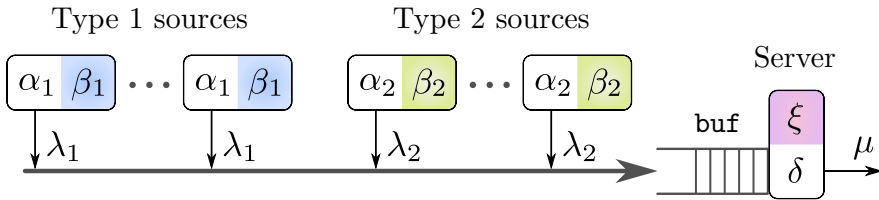


Figure 3.1: Queueing system with breakdowns

The presence of `buf` and of several distinct components, each with its own state, makes this also an attractive case for I-SPLIT. The state of the corresponding Markov process $X = \{X_t \mid t \geq 0\}$ should include the server status: an inherently Boolean random variable which can be encoded as an integer taking the value 1 when the server is operational and 0 otherwise. Process X should also include information regarding the status of each source, which can be equally encoded as integers, and of the number of packets in the buffer, of clearly integral nature. Given thus X , applying multilevel splitting in any of its flavours requires one

to decide how important each state X_t is. Equivalently: how should the importance function f be defined in this setting?

Let $buf \in \mathbb{N}$ denote the number of packets enqueued in `buf`. Given the rare event is concerned exclusively with a buffer overflow, a naïve approach would propose $f(X_t) = buf$. Let $f_1: S \rightarrow \mathbb{N}$ denote this importance function, then f_1 is oblivious of how many sources are actively sending packets to the buffer. That sounds unreasonable since no incoming packets means no possibility of overflow. Let then $f_2(X_t) \doteq buf + \sum s_1^k + \sum s_2^k$ be the importance function which adds to buf the number of active sources, with random variable s_i^k corresponding to the activation status of the k -th source of type i as described above.

The convergence times of RESTART using these two functions (and some others) were compared in [BDH15], for 95% confidence intervals with 5% relative error, testing buffer capacities $K \in \{20, 40, 80, 160\}$ with the same system parameters as in [KN99]. Surprisingly f_1 behaved at least as well as f_2 , outperforming it in several occasions. In some settings the difference was remarkable, e.g. for $K = 80$ and with a global splitting value of 5, f_1 converged more than 8 times faster than f_2 .

This unexpected behaviour could be due to the use of RESTART as splitting technique, or to the particularities of its implementation in the tool used for the test. It could also be explained on theoretic grounds, arguing the joint enqueueing rate and the service attendance rate are too fast w.r.t. the up/down times of the sources. Thus including their state in the importance generates an irrelevant layering of the state space, which would then cause fruitless computational overhead during the splitting/truncation procedures. \square

Optimal and asymptotically efficient choices of importance functions evidently require deep knowledge and some assumptions over the model. However, in spite of the Markovian nature and overall simplicity of the queueing system above, the selection of a merely good importance function proved to be a tricky issue. It is for instance unclear whether taking the state of the sources into account was mistaken altogether. It may only be that their inclusion in the importance ought to be weighed down by some scaling factor. But then which factor would yield the best results?

To make matters worse, any change in the definition of the rare event can degrade the performance of otherwise well behaved functions. Say the rare event from Example 1 is instead defined as a buffer overflow when all sources

of type 1 are active. Then importance measurement should definitely include the state of these sources, thus f_2 would behave better than f_1 (right?). So a particular definition of f needs to be built not only for every system, but also for every interesting analysis of it.

Complications strive further, since even in seemingly simple scenarios where the rare event leaves little doubt as to which system components the I-FUN should consider, concealed complexities of the system (or worse, of particular system parameters) may trick the user into a natural yet inefficient choice. That is illustrated by the following case study.

Example 2: Tandem queue.

Consider a tandem Jackson network consisting of two connected queues as depicted in Figure 3.2. Customers arrive at the first queue following a Poisson process with parameter λ . After being served by server 1 at rate μ_1 they enter the second queue, where they are attended by server 2 at rate μ_2 . After this second service customers leave the system.

Time lapses between events are exponentially distributed and independent between stations. That is, inter-arrival time is independent of the service times, and the time elapsing between two services in the first (resp. second) server is independent of the arrival times and of the service times in the second (resp. first) server. Thus the stochastic process $\{(q_1, q_2)_t \mid t \geq 0\}$, where $X_t \doteq (q_1, q_2)_t$ is the number of customers in the first and second queues at time t , is Markovian and time-homogeneous. This model has received considerable attention in the literature—see e.g. [GHSZ98, Gar00, GVOK02, VA07b, LDT07, VAVA11].

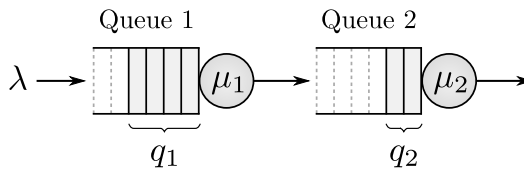


Figure 3.2: Tandem queue

For some limiting capacity $L \in \mathbb{N}$ it is interesting to study the transient behaviour of the queues for the rare event of an overflow in the second one, viz. $q_2 \geq L$. Let the system start execution with no customers in the first queue and a single one in the second queue, and let the measure of interest be the probability of full occupancy in the second queue before

it empties (this is called a *regenerative cycle* in [GHSZ99, VAVA06]). How then should f be defined?

Since the rare event involves only the second queue a naïve alternative is $f((q_1, q_2)_t) = q_2$. It would be strange however that the value of q_1 played no role at all in the true importance of X_t : even with $L - 1$ customers in the second queue, no rare event can be immediately observed if $q_1 = 0$. Name $f_1(X_t) \doteq q_2$; most modern literature discourages choosing f_1 as importance function, in view of considerations similar to these.

In contrast to state $(0, L - 1)$ as presented above, let the current system state be $(L, 3/4 L)$. Then, providing the first server is not much slower than the second one, X_t could quickly lead to the desired overflow, despite the fact that q_2 is quite far from full. This leads to propose $f_2(X_t) \doteq q_1 + q_2$ as state importance. However, if the first queue is the bottleneck and the rarity of the overflow is due to very fast service times at the second queue, the value of q_1 has little influence on the true importance of X_t , and f_2 should not perform that well.

Generalising in this direction one comes up with the family of functions $f_{\alpha_1, \alpha_2}(X_t) \doteq \alpha_1 q_1 + \alpha_2 q_2$, where the selection of the weights $\alpha_i \in [0, 1]_{\mathbb{R}}$ is behind the resulting performance of the function in each particular framework. Then $f_1 = f_{\alpha_1=0, \alpha_2=1}$ and $f_2 = f_{\alpha_1=1, \alpha_2=1}$. As it happens, the optimal choice of weights depends on the comparative order between the loads of the queues, ρ_1 and ρ_2 [VAVA06, LDT07, LLGLT09]. These loads are inversely related to the service rates following the formula $\rho_i \doteq \lambda/\mu_i$. But the essence of the question still remains: how should α_1 and α_2 be chosen in order to maximise the efficiency of function f_{α_1, α_2} ?

In a setting where $\rho_1 < \rho_2$, [VA07b, Sec. 4.1] suggests using $\alpha_1 = \alpha_2 = 1$, i.e. f_2 . The author derives this formula when looking for the linear combination between q_1 and q_2 which would minimise some expression of the variance of the estimator. An adaptation of this approach is also followed in [LDT07, Sec. 5.2]. Both works report good results: the measured variance of f_1 was quite bigger than that of f_2 .

In contrast, in a scenario where $\rho_1 > \rho_2$ (viz. the first queue is the bottleneck), [VAVA06, VA07b] suggest using $\alpha_1 = 0.6$, $\alpha_2 = 1$, as optimal weights to minimise the variance. This is at odds with the computational times reported in Table 1 of [BDH15], where f_1 was the fastest ad hoc function in a framework coherent with the one from those works. That table shows that both f_2 and $f_{\alpha_1=1, \alpha_2=2}$ were notably slower to converge

than f_1 . To discard outlier behaviour further tests were performed with the function $f_3 \doteq f_{\alpha_1=3, \alpha_2=5}$, for which the ratio $\alpha_1/\alpha_2 = 0.6$ is the same as for the optimal choice of weights mentioned. The performance measured for f_3 was comparable to that of f_2 in [BDH15], i.e. it took considerably longer than f_1 to build the desired CI. \square

There are two main conclusions to be drawn from Example 2. First, the selection of optimal (or good) weights $\alpha_i \in [0, 1]_{\mathbb{R}}$ for importance function f_{α_1, α_2} is by no means trivial. It depends at least on the comparative order of the queue loads ρ_i and may also be influenced by other parameters, e.g. the finitude of the queues, or the relationship between the queue sizes and the concrete load values (rather than just their comparative order). This in spite of the stark simplicity of the system, which has only two components with plain interaction dynamics. As described, variations in parameters of the system can lead to modifications in the overall behaviour, which deteriorate the performance of otherwise good importance functions.

Another implication of Example 2 is that though precise, theoretical analysis can be misleading if some factors slip from attention. Achieving a fully comprehensive analysis may prove hard: to derive the optimal weights for $\rho_1 > \rho_2$ in [VA07b], the author assumes a boundless first queue and some specific behaviour in the servers[†]. On top of the difference in the exact rate values λ, μ_1, μ_2 , and besides the scaling of α_1, α_2 in f_3 to imitate the optimal weights ratio $\alpha_1/\alpha_2 = 0.6$, these assumptions could be the reason behind the discrepancy between [VA07b] and the empirical results from [BDH15]. Also notice theoretical analysis was oriented to minimise the variance of the estimator, which should be proportional to the convergence wall-clock times, but could be suboptimal for a concrete implementation of a technique.

It is thus clear that finding weights α_i leading to an efficient importance function f_{α_1, α_2} in a particular setting of the tandem queue problem is a non-trivial task. The choice will depend on the specific values of the system parameters, any empirical assumption of the general behaviour (e.g. the first queue cannot overflow), and the definition of the rare event. Regarding the last remark, recall Example 2 deals with the probability of overflow in the second queue *within a regenerative cycle*. Which importance function would perform well for estimating the probability of the event $q_1 + q_2 \geq L$? And if

[†] From [VA07b, p. 153]: “we will make the following assumption: In a two-queue tandem Jackson network with loads $\rho_1 > \rho_2$, if the initial system state is $(q_1, 1)$ and \dots empty, q_1 customers will be in the second queue when the first one becomes empty.”

the interest was in studying the steady-state behaviour of the process?

On the whole, each time the system or the study angle change, new analyses are needed to come up with an efficient expression of the importance function for the application of I-SPLIT. As illustrated in the previous examples, this task is not only hard but also very limited in scope. That is why, for the effective use of splitting in real-life situations, counting with some automatic algorithm is paramount, to derive the importance function from the concrete descriptions of the system model and user query.

3.2 Deriving an importance function

I-FUN distillation can be approached from many angles, depending on the needs and intentions of the study. In this section the specific approach proposed is explained, for which the necessary formal basis is stated. The section concludes with the pseudocode of an algorithm to derive an I-FUN from a formal model and rare event specifications.

3.2.1 Objective

Two approaches are customary when giving instructions on how to select an appropriate importance function. From a rather practical point of view, it is possible to settle on some specific category of systems. Rigorous but not necessarily formal guidelines can be provided, to build a function with small variance for some determined splitting technique. This strategy is quite popular for the study of processes arising from real-life problems, e.g. queueing Jackson networks, since it provides solutions of bounded flexibility but which are easy to implement and use in the practical setting they were designed to attack. See e.g. [CAB05, VAVA13, VA14].

Instead, from a more abstract point of view the intention could be to build a function with appealing theoretic properties. Asymptotic efficiency, bounded normal approximation, and optimality (minimizing an expression of the variance of a given estimator) are typical goals. This is usually approached in an analytic fashion, formalizing rather strong restrictions on the nature of the systems covered, for which the presented strategy shows the desired properties. The main advantage of this approach is the quality of the resulting function, which will perform well, or as good as possible, regardless of the rarity of the event. See e.g. [GHSZ99, DD09, GHML11].

In this thesis the objective is to provide an automatic algorithm which,

for user provided formalisations of a time-homogeneous stochastic process X and some rare event to study, will yield an importance function on the full state space of X . To this aim the modelling formalisms from Section 2.1 are used for specifying the system models. In turn the rare event will be described as a property query matching those in Section 2.2.

The resulting *automatic importance function* is not required to be optimal or even asymptotically efficiency. The goal is to allow an effective application of multilevel splitting which outperforms the standard Monte Carlo approach from Section 2.3. Moreover, the automatic I-FUN is expected to perform at least as well as any simple and general ad hoc choice. That is, discarding solutions formally tailored for the particularities of the system under study, the function derived by the algorithm proposed should be as efficient as any alternative the user may come up with.

This last objective cannot be formally stated since the user could always “simply come up with” an optimal solution to the problem considered. Our strategy was to carry out an extensive verification, experimenting with case studies well-known in the literature about RES. This approach grants some consistent notion of the grade to which our aim was satisfied.

We have devised a tool that implements the derivation algorithm for the automatic importance function, and RESTART as I-SPLIT simulation technique. A user-defined importance function can also be fed to the tool, so the performance of several alternative functions can be compared on equal grounds. In this framework various models and definitions of the rare event have been studied, comparing the performance of the automatic I-FUN and several ad hoc proposals.

3.2.2 Formal setting

The derivation algorithm we will present performs a breadth-first search on the adjacency graph inherent to the state space of the model. Thus having only a finite number of system states will suffice to ensure termination. Also a single initial state is considered for the sake of simplicity.

In this chapter the focus will be on the finite discrete and continuous time Markov chains from Definitions 2 and 3, even though the derivation algorithm is oblivious of the memoryless property. This is motivated by the software tool developed to study the properties of the technique, and also because the DTMC and CTMC formalisms offer a well known basis which will facilitate explanations.

There are only two relevant properties which need to be distinguished

during the execution of a simulation: whether the current state is *rare*, i.e. part of the rare event, and whether it signals path truncation, i.e. if it is a *stop* state. The set of atomic propositions will thus generally be $AP = \{\text{RARE}, \text{STOP}\}$. The rare event $A \subset S$ will be composed of the states $s \in S$ for which $Lab(s) = \{\text{RARE}\}$, and in transient analysis the stopping set $B \subset S$ is identified by the label **STOP**, i.e. simulation paths will be truncated when they reach a state with that label. Since $B \cap A = \emptyset$ then $\forall s \in S. |Lab(s)| \leq 1$.

Definitions 2 and 3 give a hint of how to represent in an abstract data type the structure of a DTMC or CTMC model. They are quite alike: they only differ in the particular restrictions the elements of \mathbf{P} and \mathbf{R} must respectively satisfy. In both cases the system transitions are described in a (usually sparse) matrix of dimension S^2 , from which an adjacency graph of the states can be extracted. This is exploited by the algorithm introduced next.

3.2.3 Derivation algorithm

In Section 2.7 it was generally stated that the choice of importance function has serious performance implications in the application of multilevel splitting, whichever specific method is used. Examples 1 and 2 give some evidence of the sensitivity of the splitting approach to such choice. Overall it is clear that simulations following a splitting strategy are guided by the I-FUN, which selects the directions in which the computation effort should be intensified.

In Section 3.1 it is stated that importance functions try to approximate the true importance of the states as per Definition 11. That definition was purposefully expressed in terms of trajectories through the states: recall that for some simulation path running on the system model, the true importance of state $s \in S$ can be described as the probability of observing the rare event (i.e. visiting a *rare state* from $A \subsetneq S$) after visiting s . That probability tends to decrease with the distance, measured in number of transitions, between s and the rare event. This is crystal clear in the case of DTMC models where transitions have probability weights. The more simulation steps needed, the smaller the value of the product representing the joint probability of taking all the right transitions that lead from s to A in the fastest possible way.

Interestingly something similar happens with CTMC models. Starting from s , at each step a race condition can deviate a simulation path from the shortest route leading to A [†]. Hence the shorter such route is, the higher the

[†] This can be more rigorously stated studying the embedded Markov chain of the CTMC.

chance of reaching the rare event without detours.

This analysis suggests that for both formalisms, longer paths to A mean lower probabilities of observing the rare event. Thus in the average case for a simulation path starting from $s \in S$, distance to A (in number of transitions) and importance of s should be inversely related magnitudes. Therefore if one could track or at least conjecture the trajectories leading from s to states in A , some notion of the distance between them may be determined and used to choose an appropriate importance for s .

Certainly the actual probabilistic weights (or rates) of the transitions affect the importance of s too. Considerations involving these magnitudes will be postponed however to later stages of the strategy proposed in this chapter. The derivation of the importance function will be fully determined by the adjacency graph inherent to the transition matrix of the model.

The core idea is simple enough: starting simultaneously from all states in A , perform a backwards-reachability analysis on the transition system of the model, i.e. a BFS traversing the adjacency graph using edges with reversed directions. Each iteration of the algorithm visits a new layer of states, which are one step further from A than the states of the previous iteration. The successive layers of states visited in each iteration are labelled with decreasing importance. The pseudocode is presented in Algorithm 1, where \mathcal{M} is the system model and s_0 its initial state.

This way the length of the shortest path leading from each state into A is computed by means of a breadth-first search of complexity $\mathcal{O}(bn)$, where n is the size of the state space and b is the branching degree of the adjacency graph. Notice albeit $b \approx n$ in a worst case scenario, i.e. in case of a dense transition matrix, b is usually several orders of magnitude smaller than the total number of states.

For every state $s \in S$, its importance $f(s)$ is then computed as the inversion of its distance to the nearest rare state, where the distance between s_0 and A is the biggest one considered. In that respect notice the outer loop can finish *before* all states have been visited, as soon as s_0 is encountered. This is because in Algorithm 1, the initial system state is the one with the least importance, namely $f(s_0) \doteq 0$. The description of RESTART implies this must indeed be the case, since there are no D_0 events to truncate the main trial. In general having states to which f assigns less importance than s_0 can yield little benefit. Splitting for oversampling in regions so far away from the rare event is deemed to incur in unnecessary computation overhead, with little or no reward to show in return.

Algorithm 1 makes two major assumptions on its inputs:

Algorithm 1 Importance function derivation from a system model.

Input: module \mathcal{M}

Input: rare state set $A \neq \emptyset$

```

 $g(A) \leftarrow 0$ 
queue.push(A) {marks states in  $A$  as visited}
repeat
   $s \leftarrow \text{queue.pop}()$ 
  for all  $s' \in \mathcal{M}.\text{predecessors}(s)$  do
    if  $s'$  not visited then
       $g(s') \leftarrow g(s) + 1$ 
      queue.push(s') {marks  $s'$  as visited}
    end if
  end for
until queue.is_empty() or  $s_0$  visited
 $g(s) \leftarrow g(s_0)$  for every non visited state  $s$ 
 $f(s) \leftarrow g(s_0) - g(s)$  for every state  $s$ 

```

Output: importance function $f: S \rightarrow \mathbb{N}$

1. it expects to have a black-box access to the (reversed) adjacency graph of \mathcal{M} by means of the function $\mathcal{M}.\text{predecessors}: S \rightarrow 2^S$;
2. it expects to be provided the rare set $A \subseteq S$ as input.

Assumption 1 can be easily achieved, since the dynamics of finite discrete and continuous time Markov chains are usually stored in transition matrices, from which the adjacency graph can be straightforwardly obtained. Assumption 2 is less direct since the user input in that respect is a property query like those of Equations (7) and (8). Some mechanism must then be provided, to turn the logical formula expressing the rare event into a set of satisfying states. This is covered in Section 3.3.2. With these considerations in mind a proof of termination can be given.

Proposition 6 (Termination of the importance function derivation algorithm). *Let \mathcal{M} be a finite DTMC or CTMC model, and $A \neq \emptyset$ the set of rare states from the state space S of \mathcal{M} , derived from some transient or steady-state user query. Then, starting from inputs \mathcal{M} and A , Algorithm 1 terminates in a finite number of iterations.*

Proof. Since \mathcal{M} is a finite DTMC or CTMC model, the set S is finite and so is the adjacency graph derived from the \mathbf{P} or \mathbf{R} transition matrix. Dealing with a finite adjacency graph ensures the inner **for–do** loop will terminate every time, since there is a finite number of predecessors to every state. Denote by *visit* the action of pushing a state into the **queue**. The conditional statement inside the inner loop ensures every state is visited at most once, and the outer loop extracts an element from the **queue** on each iteration. Thus a finite S suffices to guarantee the first condition in the guard of the outer **repeat–until** loop will be eventually satisfied. Since this condition is a disjunction, the argument given ensures the outer loop will run a finite number of iterations. \square

As a matter of fact Proposition 6 can be applied to any finite stochastic process model \mathcal{M} , since the memoryless property of the Markov chains was not used in the proof. The only complication could arise from the way in which the system transitions are expressed, but as long as the black-box function $\mathcal{M}.\text{predecessors}: S \rightarrow 2^S$ is provided, the derivation algorithm will terminate in a finite number of steps.

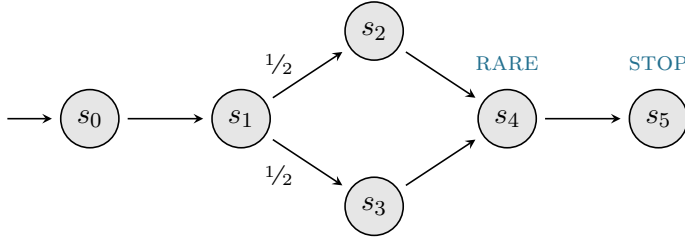
Algorithm 1 yields a function so that every simulation following a shortest path from the current state to the rare set, will traverse a monotonically increasing sequence of importance values. Call this the *monotonicity condition* on the I-FUN. Nevertheless it must be highlighted that the function is not necessarily *correct*, in the sense that it will not always yield the true importance of the states of the model. That is evident since the weights or rates of the transitions are disregarded. Take for instance a DTMC where states s and s' are respectively two and one transitions away from A , but where both transitions linking s to A have probability 1, whereas the one linking s' to A has probability $1/2$. Algorithm 1 will give a higher importance to s' , which is clearly at odds with Definition 11 of true importance.

This issue can be regarded as a pitfall of the algorithm, but is in fact effectively countered in the comprehensive approach introduced in the next section, which automates the application of the whole importance splitting procedure. This exemplifies a major benefit of using all-embracing strategies to attack a problem like RES: the push-button approach we seek is certainly convenient from a practical point of view; but more importantly, it can balance weaknesses and strengths among the several steps involved in the process. On top of that, automating I-SPLIT is a way to avoid mistakes derived from misinterpretations of the subtleties of each particular splitting technique. An erroneous implementation combined with a poor selection

of importance function can sometimes yield incorrect estimations, as in the next example.

Example 3: Misestimation of an incorrect RESTART.

Consider the six-state DTMC depicted below and the importance function $\{(s_0, 0), (s_1, 1), (s_2, 1), (s_3, 0), (s_4, 2), (s_5, 0)\}$ on it. The initial state is s_0 and the (not actually) rare event is $A = \{s_4\}$. The importance regions are given by the single threshold $L_1 = 1$, i.e. zones $Z_0 \doteq \{s_0, s_3, s_5\}$ and $Z_1 \doteq \{s_1, s_2, s_4\}$ form the states partition to be used by multilevel splitting.



Given s_5 is the stopping state, the transient probability of a simulation path from s_0 to reach the rare event before stopping is trivially $\gamma = 1$, as also a standard Monte Carlo analysis by simulation would suggest.

Recall the RESTART estimator for transient analysis is $\hat{\gamma} = \frac{M}{K N_0}$, where M is the number of paths reaching the rare event, N_0 is the total number of RESTART simulations started from s_0 , and K stands for the stacked up splitting factor between the importance of the initial system state and the max importance value.

Since there is a single threshold, L_1 , K equals the splitting performed at L_1 . Also it is reasonable to assume $K > 1$, otherwise RESTART would not differ from standard Monte Carlo; say e.g. $K = 2$.

Notice that any simulation taking the path through s_3 will suffer a truncation of all offsprings created in s_1 , since they move from zone Z_1 into Z_0 causing a D_1 event. Only the original trial from level 0 is able to survive the $s_1 \rightarrow s_3$ transition. In the next simulation step, when this trial takes the $s_3 \rightarrow s_4$ transition, it moves into the rare set A and *simultaneously* triggers a B_1 event.

There are two possible implementations of RESTART at this point: one option is to attend the B_1 event first; the other option is to consider first the entrance into the rare set A . In the latter case no re-splitting is done when the simulation path moves from Z_0 into Z_1 , and then statistically we would have

$$\lim_{N_0 \rightarrow \infty} M = \frac{3}{4} N_0,$$

because half of the simulation paths would take the $s_1 \rightarrow s_3$ transition, and for $K = 2$ half of those paths will be offsprings which get truncated upon visiting s_3 .

Thus applying RESTART in the way described results in the erroneous estimate $\hat{\gamma} \approx \frac{3/4 N_0}{2 N_0} = 3/8 \neq \gamma$. Notice that the gap between $\hat{\gamma}$ and the real transient probability γ is exacerbated by increasing the splitting value. \square

There are two issues whose conjunction lead to a wrong RESTART estimate of $\mathbb{P}(\neg \text{STOP} \cup \text{RARE})$ in Example 3. First, some simulation paths moved from Z_1 into the lower importance zone Z_0 , despite they were getting closer to the rare event (in particular the monotonicity condition does not hold for the importance function proposed). Second, the implementation of RESTART attended entrance into A before splitting by the B_1 event.

In an ad hoc approach, still assuming a perfect implementation of the splitting technique of choice, there remains the issue of the importance function. Merely using an I-FUN without the monotonicity condition does not suffice to produce the wrong estimate in Example 3. Even so, any function not preserving such condition may incur in high inefficiencies. That is shown in the example above in the splitting-truncation-resplitting of a simulation following the $s_1 \rightarrow s_3 \rightarrow s_4$ trajectory. Theoretically this is quantified for RESTART in the f_V factor from Section 2.7.

In simple systems like the above it seems trivial to ensure the monotonicity condition by any ad hoc proposal. Yet the complexities of the model and definition of the rare event in real life situations tend to complicate matters. For instance there are system where a failure can be triggered by different configurations of the components, not necessarily related among them. In such situations splitting may take advantage of layering the state space in a way where the rare event is not only at the highest importance value.

Automatic techniques like Algorithm 1 guarantee the resulting importance function will have the desired properties. Properly embedded in the

comprehensive approach of the following section, this can avoid estimation issues like the one illustrated in Example 3.

3.3 Implementing automatic I-SPLIT

Counting with an algorithm to derive the importance function is a first major step in the direction of automating the application of multilevel splitting, yet it does not suffice. With few exception these techniques require also to choose the number of thresholds and their exact importance value. Moreover fixed splitting approaches need a selection of the split to perform at each threshold, and fixed effort needs an analogous selection of the effort to dedicate on each importance level.

This section presents another contribution of the thesis: an automatable strategy to apply I-SPLIT for system analysis by RES. This includes a framework for system modelling and specification of the user query, an algorithm to select the thresholds, and automatable execution of simulations using splitting techniques to estimate an answer to the query.

3.3.1 Modelling language

In spite of their mathematical accuracy, Definitions 2 and 3 of finite Markov chains fail to provide a user-friendly formalism for describing probabilistic/stochastic processes. That is because an explicit specification of S is impractical. Realistic models can easily have thousands of different configurations which would need to be represented as the set S of opaque states [Har15].

A typical solution is adding an abstraction layer by means of *typed system variables*. For the scope of this thesis it suffices to consider integral variables, where Booleans can be encoded as $\{0, 1\}$ -valued integers (though for notational purposes we might use the symbols \top and \perp to denote the logical *true* and *false* respectively), and disregarding variables which take values from dense sets. In such setting, the set of all possible valuations of the variables considered conforms the state space of the Markov chain. To preserve finitude these variables need to be granted a bounded number of possible values.

Definition 13 (Symbolic states). Let $\{v_i\}_{i=1}^m$ be a sequence of bounded integral variables, i.e. each v_i can take values from some finite non-empty set $V_i \subsetneq \mathbb{Z}$.

Then a vector $(v_1, v_2, \dots, v_m) \in \mathbb{Z}^m$ of specific valuations of the variables will be called a *symbolic state*.

Definition 14. Given a bounded integral variable v taking values from V , the *range of v* is $\#v \doteq |V|$, i.e. the number of different values v can take.

Definition 15 (Concrete states). Say the state space S of some finite (discrete or continuous time) Markov chain \mathcal{M} corresponds to the set of all possible valuations of the bounded integral variables $\{v_i\}_{i=1}^m$. Then each state $s \in S = \{s^j\}_{j=1}^M$ will be referred to as a *concrete state* of \mathcal{M} , where M is the amount of feasible distinct valuations, viz. $M = \prod_{i=1}^m \#v_i$.

Notice a symbolic state is an m -dimensional vector expressed in terms of variables, and a concrete state lies in the (flat) finite set S . Definitions 13 to 15 establish an implicit bijection between the *symbolic state space* of a sequence of variables and the *concrete state space* of the Markov chain \mathcal{M} for which they were defined. It will be said that \mathcal{M} *is bound to variables* $\{v_i\}_{i=1}^m$ and vice versa.

In Examples 1 and 2 from Section 3.1 the rare and stopping events, viz. the states sets A and B , were defined in terms of some system component, namely an overflow in a queue. This naturally speaks of a particular valuation of the variable representing the number of customers in the queue. However, sets A and B must be declared in terms of the atomic propositions AP and the labelling function Lab defined on top of S . Clearly a more practical and variable-related way of expressing them is desirable.

In general and unless noted otherwise, we will assume that the rare and stopping sets will be declared by the user as symbolic states. The bijection established by the previous definitions ensures this defines unique A and B subsets of concrete states in S . For instance in the tandem queue example where the rare event was defined as $q_2 \geq L$, the concrete states labelled with **RARE** will be those corresponding to all symbolic states where the variable q_2 has a valuation equal to or greater than L .

There remains the issue of the model transitions. Some efficient abstract data type can be used to represent the sparse matrix corresponding to the probabilities in \mathbf{P} or the rates in \mathbf{R} from Definitions 2 and 3. Typical choices are CSR and CSC sparse representations, and MTBDD. Representation efficiency notwithstanding, it is impractical to request such input directly from the user. Even in very sparse cases the size of the matrix will most likely be too large to resort to an explicit declaration.

Several abstract languages have been devised to specify the dynamics of a process. These typically allow to speak directly in terms of symbolic states,

i.e. of certain variables valuations. *Edges* are then defined at the abstraction level of variables, each corresponding to one or more transitions at the lower level of concrete states.

Definition 16 (Edges). For Markov chain \mathcal{M} bond to variables $\{v_i\}_{i=1}^m$, let pre be a Boolean condition on the variables. Also for $k \leq m$ and indices i_ℓ taking disjoint values in $\{1, \dots, m\}$, consider arithmetic expressions $\{ex_{i_\ell}\}_{\ell=1}^k$ involving these variables. Assume that expression ex_{i_ℓ} results in valid values for variable v_{i_ℓ} , and denote $pos \doteq \bigwedge_{\ell=1}^k (ex_{i_\ell})$. Then $pre \rightarrow pos$ is an *edge* on \mathcal{M} , where pre is the *precondition* or *guard* of the edge, and pos is the *postcondition* or set of *actions* of the edge.

Intuitively, the guard of an edge tells when its actions can be applied. Notice several transitions at the level of S can be covered by a single edge, since a guard can speak of a range of valuations. For instance the edge

$$(0 < q_1 < L \wedge arrival) \rightarrow (q_1 + 1)_{q_1}$$

represents transitions whose originating state corresponds to the variables valuation $arrival \equiv \top$ and $q_1 \in \{1, \dots, L - 1\}$. For $L > 2$ that is strictly more than one transition at the level of S . The sub-index decorating the postcondition is used above to signify that the value resulting from the expression $q_1 + 1$, for the current value of q_1 , will be applied to variable q_1 .

To describe a DTMC or CTMC model, Definition 16 cannot be used as it was given because it lacks a key component: the probabilistic weights of the transitions from \mathbf{P} and the transition rates from \mathbf{R} . This is covered in different ways by the modelling languages available in the literature. Here the PRISM language is adopted since: it has a clear and relatively simple syntax; it is a de facto standard in the field of probabilistic model checking; and the tool implementing the derivation algorithm from the previous section was developed as a modular extension of the PRISM tool.

Code 3.1: PRISM syntax example for a DTMC

```

1 dtmc
2 const double p = 0.4;
3 const double q = 0.001;
4 module Light
5     is_on: bool init false;
6     [] !is_on -> ( p): (is_on'=true)
7                 + (1-p): true;           // i.e. do nothing
8     [] is_on -> ( q): (is_on'=false)
9                 + (1-q): true;
10 endmodule

```

An extensive description of the syntax of the PRISM language can be found in the “Manual” section of its webpage. A toy example is shown in Code 3.1 which models a light switch in a discrete-time environment. When it is off the light has probability p of being turned on. Conversely the light is turned off with probability q when it is on. The semantics attached to the PRISM language syntax in terms of DTMC and CTMC can be found in David A. Parker’s Ph.D. thesis, [Par02].

From now on, references to the *concrete level* of a Markov chain will be implicitly speaking of S , \mathbf{P} or \mathbf{R} , and transitions between concrete states. Conversely, references to the *symbolic level* will be referring to the variables and edges of some high-level description (say, using the PRISM input language) of the Markov chain. So for instance the symbolic level of the `Light` model from Code 3.1 involves e.g. the variable `is_on` and the edges from lines 6 to 9, whereas the concrete level refers to the DTMC which gives semantics to this PRISM model.

Some implementation decisions shape the way in which DTMC and CTMC models are expressed in PRISM. Examples are: how overlapping guards in several edges are interpreted; the way in which parallel execution is handled when the model is composed of many modules; and how to synchronise the execution of such modules. All details relevant for this thesis are quickly reviewed in the next practical examples.

Example 4: Single queue DTMC model.

Consider a buffered server operating in a discrete time environment. At each *time tick* the system will receive a new packet with probability `lambda`, enqueueing it in the buffer `q1`. In turn, when the queue is not empty, at each time tick the server will process and dequeue a single packet with probability `mu1`. Notice arrival and processing could happen simultaneously during the same tick, resulting in an unchanged state of the queue once the time tick has elapsed. A model of this process is expressed next using the PRISM language.

Code 3.2: PRISM discrete queue model

```

1 dtmc
2
3 const int c = 12;           // Capacity of the queue
4 const double lambda = 0.1; // Probability of packet arrival
5 const double mu1 = 0.14;   // Probability of packet processing
6

```

```

7  module DiscreteQueue
8
9      q1: [0..c] init 1;
10
11     [] (q1=0) -> ( lambda): (q1'=q1+1)
12                + (1-lambda): true;
13     [] (0<q1 & q1<c) -> ( (lambda)*( mu1)): true
14                        + ( (lambda)*(1-mu1)): (q1'=q1+1)
15                        + ((1-lambda)*( mu1)): (q1'=q1-1)
16                        + ((1-lambda)*(1-mu1)): true;
17     [] (q1=c) -> ( (1-lambda)*mu1): (q1'=q1-1)
18                + (1-(1-lambda)*mu1): true;
19  endmodule

```

Observe how each possible valuation of variable `q1` is treated in the three edges. The three corresponding guards form a partition of the state space. If some value was not covered, it would result in unspecified behaviour (a *deadlock*), since the model could not take any action upon reaching such valuation. If instead two guards overlap, the model would show nondeterministic behaviour since two potentially different actions (i.e. two different transitions) could be performed from the same system state. The PRISM tool recognises these undesired situations and warns the user about their presence in the model.

Though simple, this model can be analysed for rare behaviour. For instance one could study the probability of reaching the maximum queue capacity, starting from a non-empty state, before all packets in the queue get processed by the server. This transient analysis can be forced into the rare event scope simply by fiddling with the probabilities `lambda` and `mu1` and the queue capacity `c`. \square

Each edge in a PRISM DTMC model specifies all the transitions outgoing the states which satisfy its guard. Since the transitions outgoing a state in a DTMC are probabilistic, their weights must add up to one. As mentioned in Example 4, when two guards from two edges overlap, i.e. if both can become true for some valuation of the variables, nondeterminism arises. Specifically, PRISM interprets such situation as two disjoint sets of probabilistic transitions outgoing the states which satisfy these guards. A simulation path would have to choose between those sets, with no hint regarding which of them to follow. This is nondeterministic behaviour, and falls outside the scope of the DTMC formalism. Summarizing, a PRISM DTMC model *cannot have edges whose guards overlap*.

The situation is different in the stochastic scenario, since race conditions

naturally express the existence of several unrestricted (other than having a positive rate) transitions leaving a state. Therefore, a PRISM CTMC model *can have edges whose guards overlap*. The concrete states satisfying multiple guards would simply resolve the resulting race condition, performing the transition which fired first and discarding the others.

As defined so far, a Markov chain models the sequential evolution of a probabilistic or stochastic process. In reality, however, most hard- and software systems are not sequential but parallel in nature [BK08, sic]. A process can be defined by the parallel execution of its components, also called *modules*. Notice the `module` and `endmodule` keywords in Codes 3.1 and 3.2. The PRISM language allows the definition of several modules, and the process resulting from the parallel execution of all of them is referred to as the *global system model*.

The individual modules can be totally independent during the parallel execution of the global system model, evolving autonomously, or can communicate and cooperate in some way. The first option is called *interleaving* and for the second option there are many alternatives, like shared variables and channel systems [BK08]. A broadcast variant of the *handshaking mechanism* will be used along the thesis, where modules execute certain transitions *synchronously* and interleave execution of all other transitions.

Handshaking introduces a new type of element, *synchronisation labels* or *actions*, used by the modules to communicate among them. These actions are the means by which parallel components take a transition in synchrony. They are closely related to the actions set A from Definitions 1 and 4 of LTS and SA, though here they will be used to synchronise Markov chains.

In the PRISM language each edge in a module is labelled with an action wrapped in square brackets. If the brackets are empty, the special τ (tau) transition is assumed, which does not synchronise and implies an *interleaving edge*. That is the case for all edges in the PRISM model from Code 3.2. Oppositely, when two or more edges from different modules share a label different from τ , they must be executed synchronously or not at all. This is illustrated in the following example.

Example 5: Tandem queue CTMC model.

Recall the Markovian tandem queue from Example 2. It was introduced in a continuous time setting and can henceforth be modelled as a CTMC; Code 3.3 is a PRISM model of it. The system is represented as three de-

pendent modules, `Arrivals`, `Queue1`, and `Queue2`, which run parallelly and synchronise through the action labels `arrival`, `service1`, and `service2`.

Code 3.3: PRISM tandem queue model

```

1  ctmc
2
3  const int    c = 8; // Capacity of both queues
4  const int lambda = 3; // rate(-> q1    )
5  const int  mu1 = 2; // rate(  q1 -> q2  )
6  const int  mu2 = 6; // rate(          q2 ->)
7
8  module Arrivals
9      // External packet arrival
10     [arrival] true -> lambda: true;
11 endmodule
12
13 module Queue1
14     q1: [0..c-1] init 0;
15     // Packet arrival
16     [arrival] q1<c-1 -> 1: (q1'=q1+1);
17     [arrival] q1=c-1 -> 1: true;
18     // Packet processing
19     [service1] q1>0 -> mu1: (q1'=q1-1);
20 endmodule
21
22 module Queue2
23     q2: [0..c-1] init 1;
24     lost: bool init false;
25     // Packet arrival
26     [service1] q2<c-1 -> 1: (q2'=q2+1);
27     [service1] q2=c-1 -> 1: (lost'=true);
28     // Packet processing
29     [service2] q2>0 -> mu2: (q2'=q2-1);
30 endmodule

```

Consider the external arrival modelled in the module `Arrivals`, line 10. For the arrival to happen, action `arrival` is broadcast for synchronisation with the other modules. Since `Queue2` only reacts to actions `service1` and `service2`, it ignores the issue altogether. Module `Queue1` however does synchronise on the `arrival` action, so the arrival will be allowed iff one of its edges in lines 16 and 17 is enabled. The guards of these edges were chosen so that at all times, exactly one of them will be enabled. Hence the external arrival can always take place, which is consistent with a realistic queueing system.

In this way, the packet arrival is modelled via a synchronous execution of the corresponding `arrival`-edges in both modules. An analogous mechanism is set in motion when the server in the first queue processes

a packet. Then the action is `service1` and the modules participating in the synchronous transition are `Queue1` and `Queue2`. No synchronisation uses action `service2`; it could thus be replaced with τ , i.e. changing `[service2]` for `[]` in line 29, without affecting the global behaviour.

This Markovian model of a tandem queue presents several opportunities for the study of rare behaviour. From the transient point of view, it is interesting to know which is the probability of losing a packet in the second queue due to an overflow (indicated by `lost'=true`), before the server processes all packets and `q1` evaluates to zero. From the steady-state point of view one can be interested in the long run probability of observing such packet loss. The rarity of these events can be tuned with the system parameters expressed as constant integers in Code 3.3.

As last remark notice that in the way they are presented and disregarding the indicator variable `lost`, modules `Queue1` and `Queue2` are exact copies modulo renaming of variables and synchronisation actions. More queues could be added in the same fashion, extending the model to an n -queues tandem Jackson network for arbitrary $n \in \mathbb{N}$. \square

In the PRISM language an action can also be blocking, when one of the modules taking place in the synchronisation does not have an enabled guard. Say for instance line 17 from Code 3.3 (in module `Queue1`) is removed. When `q1=c-1` no arrival would then be allowed, since `Queue1` synchronises on label `arrival` but the single `arrival`-edge left would have a disabled guard. Thus transitions in the module `Arrivals` would be blocked.

This inter-module synchronisation scheme also allows race conditions at the level of the the global system model, but only in interleaving transitions. When two guards from different modules are enabled and the edges do not synchronise, a race condition is formed and resolved. The values sampled from the exponential distributions involved will determine which one takes place, just like in the intra-module case

To illustrate the above, consider the initial state of the tandem queue from Example 5. Notice the single guard in the `Arrivals` module is always satisfied. Also, since initially `q2` is assigned the value `1` in `Queue2`, the last edge of that module (line 29) is enabled as well. Hence two transitions are enabled in the initial global state of the system model: one corresponding to an arrival in module `Arrivals`, and another one corresponding to a packet processing in module `Queue2`. That is a race condition at global scope since both edges do not synchronise.

A last important matter to consider about the PRISM language is the resulting rate of synchronising transitions. For interleaving edges executed without synchronisation, the rate of the underlying transitions at concrete level is the rational number preceding the colon in the edge. That is e.g. the case of `mu2` in line 29 of Code 3.3.

When instead several edges from different modules are executed synchronously, *the product of their rates* will be the rate of the underlying transition in the global system model. Like in Example 5, this is commonly dealt with by setting the desired rate of the global transition in the edge of a single module. All the synchronising edges of all the other modules are then given rate 1, so the resulting product will be the desired global rate.

3.3.2 User query specification

Recall there are two angles from which rare events are studied in this thesis: transient and steady-state analysis. Equations (7) and (8) in Section 2.5.1 already provide formulae from temporal logics to express queries regarding respectively transient and steady-state behaviour.

Yet those expressions assume an explicit representation of the rare and stopping sets of states, A and B , or at best in terms of their characterizing sets of atomic propositions. Given that A and B can be defined at symbolic level, it is also reasonable to expect that expressions involving variables, and not atomic propositions or concrete states, will be the means to query the probability values sought.

The property specification language of the PRISM tool subsumes several probabilistic temporal logics, including PCTL and CSL. Besides, at top level it offers the quantitative operators $P=?$ and $S=?$ related to those logics, which respectively yield a numerical value regarding behaviour of transient and steady-state nature. Namely, these operators return “the probability” of the set of states which satisfy the succeeding subformula, thus fitting smoothly in the current framework.

To perform transient analysis, the user will specify the query

$$P=? [\neg stop \ U \ rare]$$

where both *stop* and *rare* are Boolean expressions involving literals, constants, and variables defined in the PRISM model of the system. Expression *stop* identifies the stopping states, which truncate the simulation paths reaching them. Expression *rare* identifies the rare event of interest.

Steady-state analysis is instead requested with the query

$$S=? [\textit{rare}]$$

for the same definition of *rare*. Interestingly, even though CSL was designed to study systems in a continuous time environment, the long run query above can be also used with DTMC models. It will yield the probability of visiting the states satisfying *rare*, once the system is in equilibrium.

Example 6: Rare event user queries for the tandem queue.

Resuming the study of the tandem queue, suppose the user wants to know how likely is it to lose a packet in `Queue2` before it empties. Such loss involves the server of the first queue processing and sending a packet to a fully occupied second queue, viz. when $q2=c-1$. Making use of the indicator variable `lost`, the property query to perform the corresponding transient analysis is:

$$P=? [q2>0 \text{ U } \textit{lost}],$$

which formally asks for “the probability of not observing an empty second queue, until a packet is lost in `Queue2`.”

Recall that initially there is one packet stored in `q2`, which is necessary to keep the initial state out of the stopping set B . Practically this means that without the “`init 1`” directive in the definition of `q2` (line 23 of Code 3.3), all simulations would stop as soon as they begin.

If instead the user is interested in the long run probability of losing a packet in `Queue2`, the query should be:

$$S=? [\textit{lost}].$$

Since the tandem queue as described and modelled in Examples 2 and 5 is an ergodic system[†], this property is in fact oblivious of the initial value of `q2`. □

3.3.3 Selection of the thresholds

The two previous sections provide the framework to specify the model and the rare event queries. All complies with the PRISM input language, and

[†] The underlying Markov chain is ergodic because all its states are aperiodic and positive recurrent—see e.g. [Tso92].

this tool has a built-in simulation engine, so analysis by the standard Monte Carlo approach from Section 2.3.2 could be directly performed.

Using importance splitting is not so direct, due to the extra information this technique requires. Section 3.2.3 explains how to automatically derive the importance function from the user input just described, but that is not all. Most splitting strategies require choosing the threshold values, at which either path-cloning will take place (e.g. for fixed splitting), or each stage of the simulation starts and ends (e.g. in fixed effort).

One could simply use each importance value as a threshold, e.g. splitting each time a simulation visits a state with higher importance than the previous one. Obviously this has big chances of incurring in a large computational overhead, which could easily render useless any gain derived from the use of splitting. Choosing every other importance value as a thresholds sounds more reasonable, but it is again nothing else than a blind choice.

We acknowledge two solutions to this problem: selecting the thresholds ad hoc, tailored for the specific system under study, or using an algorithm to analyse (e.g.) the automatic I-FUN produced by Algorithm 1, trying to *derive the thresholds* as well. In general and following the automatable approach of the whole thesis, it is desirable to choose the thresholds adaptively, considering the structure of each particular model.

From the popular I-SPLIT implementations mentioned at the end of Section 2.5.1, Adaptive Multilevel Splitting stands out due to its “dynamic thresholds discovery.” Recall this technique and its successor, Sequential Monte Carlo, run simulations on a system where only the importance of the states is known. By means of a statistical analysis of the maximum importance reached by each simulation path, the values of the thresholds (or its analogous in that setting) are incrementally discovered from the initial state up to the rare event.

Hence, to carry out the desired fully automatic application of I-SPLIT the following implementation options arise:

1. using Adaptive Multilevel Splitting (or Sequential Monte Carlo) as standard splitting technique, discarding the need of thresholds altogether;
2. extracting the algorithmic idea of these approaches to find the thresholds, and use them with any other splitting technique.

RESTART was selected as default splitting algorithm due to its nice practical properties—see Section 2.6. Moreover, both Adaptive Multilevel Splitting and Sequential Monte Carlo would need to be generalised (if possible)

if we wish to perform steady-state analyses. Therefore we chose the second approach; the pseudocode for the selection of the importance values to use as thresholds is presented in Algorithm 2. It takes the following parameters:

- \mathcal{M} - the system model;
- f - the importance function;
- n - the number of independent simulations launched per iteration;
- k - the number of *successful* simulations among the n launched;
- m - the number of discrete events to generate for each simulation.

Besides, the algorithm uses the following internal variables:

- `sim` - an array to store states with the max importance of each simulation;
- `T` - a queue to store the importance values selected as thresholds.

Routine $\mathcal{M}.\text{simulate_ams}(s, n, m, f, \text{sim})$ in Algorithm 2 launches n independent simulations from state s , and stores in the array variable `sim` the states embodying the maximum importance values observed in each simulation (n in total). Sorting these states in increasing order according to their importance value leaves in the $(n - k)$ -th position of `sim` the state embodying the $(n - k)$ -th *n-importance*-quantile, which may become a new threshold.

The numerical inputs of the algorithm, $k, n, m \in \mathbb{N}$, must be selected ad hoc by the user. Heuristics based on the nature of the importance function f and the model \mathcal{M} are easy to implement and have been used in the software tool developed. According to our empirical observations, as long as the bounds $m \in [10^3, 10^5]$, $n \in [10^2, 10^4]$, and $k \approx n/s_m$ are respected, where s_m is the maximum splitting value on any threshold, those values only have a moderate impact on the efficiency of the algorithm.

This *adaptive selection of thresholds* (plus the splitting factors) provides all complementary information needed by an I-FUN, ad hoc or automatic. In particular, Algorithm 1 exploits the structure of the adjacency graph of a model \mathcal{M} to derive the function, but disregards its probabilistic/stochastic nature. The paths generated by $\mathcal{M}.\text{simulate_ams}(\dots)$ in Algorithm 2 do consider such information, in a state space already labelled with importance. Thus the resulting thresholds, which are the most important metadata used by RESTART during simulations, reflect the full behaviour of the model.

There are a few caveats with the use of this approach in the setting of the thesis, related to the way in which Adaptive Multilevel Splitting was introduced by Cérou et al. in [CG07]. They are discussed next.

Algorithm 2 Selection of thresholds with Adaptive Multilevel Splitting.

Input: module \mathcal{M}

Input: importance function $f: S \rightarrow \mathbb{N}$

Input: simulations setup $k, n, m \in \mathbb{N}_{>0}$, $k < n$

```

Var:  $\text{sim}[n]$    Type: array of states
Var:  $T$          Type: queue of integers           {the thresholds}

 $s \leftarrow \mathcal{M}.\text{initial\_state}()$ 
 $T.\text{push}(f(s))$ 
 $\text{fail} \leftarrow \perp$ 
repeat
   $\mathcal{M}.\text{simulate\_ams}(s, n, m, f, \text{sim})$ 
   $\text{sort}(\text{sim}, f)$ 
   $s \leftarrow \text{sim}[n - k]$ 
  if  $T.\text{back}() < f(s)$  then                               { $T.\text{back}()$  does not dequeue}
     $T.\text{push}(f(s))$                                          {new threshold:  $(n - k)$ -th  $n$ -quantile}
  else
     $\text{fail} \leftarrow T$ 
  end if
until  $T.\text{back}() = \max(f)$  or  $\text{fail}$ 
for  $i \leftarrow T.\text{back}() + 1$  to  $\max(f)$  do
   $T.\text{push}(i)$                                            {unreached importance values become thresholds}
end for

```

Output: queue with threshold values T

Continuous vs. discrete spaces

The proof that Adaptive Multilevel Splitting yields an optimal estimator is based on a continuous state space and importance function on them [CG07]. This means that in the original algorithm, thresholds can be chosen arbitrarily close to each other, which is one of the hypotheses used to ensure optimality.

In the scope of this thesis state spaces are finite, and even though optimality is not a major concern, the continuity hypothesis of Adaptive Multilevel Splitting may have repercussions regarding termination of the idea behind Algorithm 2. More precisely, simulation paths tend to go down—in terms of importance—as the rarity of the states increases. In particular it could happen that most or even all simulations launched by $\mathcal{M}.\text{simulate_ams}(\dots)$

visit states whose importance is strictly below the importance of the starting point at state s .

If more than $n - k$ simulations go down in the way described above, the $(n - k)$ -th n -quantile from `sim` will not be above the previously defined threshold. An iteration of the **repeat–until** loop in Algorithm 2 would then fail to provide a new value to store in `T`.

To remedy such situations it was decided to consider all unreached importance values as thresholds. This strategy is coherent: if that many simulations go down without visiting higher importance states, then they are dwelling in a very rare zone, where the chances of observing the next importance value are less than k/n . Thus the best that can be done is to regard such next importance value as a threshold. Notice this is exacerbated by the discreteness of the states and the importance function.

In Algorithm 2 the Boolean variable `fail` is used to identify and defuse the cases when the $(n - k)$ -th n -quantile does not yield a higher threshold value. This provides enough conditions to prove termination.

Proposition 7 (Termination of the thresholds selection algorithm). *Let \mathcal{M} be a finite DTMC or CTMC model, f an importance function with image on the natural numbers, and $k, n, m \in \mathbb{N}, k < n$. Then, from those inputs, Algorithm 2 terminates after executing a finite number of instructions.*

Proof. The final **for–do** loop has a finite range and a single constant-time instruction in its body. Therefore it will terminate, and it suffices to prove termination of the main **repeat–until** loop. Each of the n simulations launched by the `\mathcal{M} .simulate_ams(\dots)` routine generates m discrete events and finishes. The sorting routine also performs its task using a finite amount of instructions. Thus it only remains to prove the guard of the loop is satisfied in a finite number of steps. If an iteration does not yield a state whose importance is higher than the last selected threshold, `fail` is set to \top and the guard of the loop is satisfied. Otherwise the conditional inside the loop ensures every call to `\mathcal{M} .simulate_ams(\dots)` in every iteration of the loop yields a state with an importance higher than the last value stored in `T`. Since S is finite, so is the codomain of f , and therefore the number of distinct values that can be considered in this way is finite. Hence, in a finite number of iterations the value $\max(f)$ will be stored in `T`, and the guard of the loop will be satisfied. \square

Just like with Proposition 6 for the I-FUN derivation algorithm, Proposition 7 is oblivious of the memoryless property. That means Algorithm 2 can

actually be used with any time-homogeneous stochastic process.

In spite of its coherence, the termination strategy used in Algorithm 2 is quite harsh. There are milder alternatives to the Boolean variable `fail`, like using a counter of failures and a predefined tolerance bound. Each time the condition `T.back() < f(sim[k])` fails, the loop would be repeated for the same `T` and `s`, but incrementing the failures counter and the effort of `M.simulate_ams(...)` (e.g. increasing `m` or `n`). If the counter reaches the tolerance bound, then the **repeat–until** loop is broken as in Algorithm 2. On the other hand, if a new threshold is found, the counter and any other modified variables such as `m` or `n` can be reset to their original values.

The importance of the rare event

Another issue in the current setting w.r.t. the original theory from [CG07] is the importance value of the rare states. In Adaptive Multilevel Splitting the rare event is defined as the states of a strong Markov process above certain barrier M , and the algorithm attempts to reach these states. Such setting coincides with the one from Section 2.5.1 where $A = E_n$, i.e. when f maps all rare states to the highest importance values, which is the case of the automatic I-FUN. However some systems do not fit naturally in this characterization, as it will be shown in Section 3.6.

An easy workaround is to aim at reaching the maximum importance value instead of reaching a rare state. That is exactly the approach of Algorithm 2. There is no check considering the user definition of the rare event; all that matters is the importance distribution of f .

The drawback of this approach is that in some cases, $\max(f)$ represents an extremely rare situation, and the “more common” rare events are represented by states whose importance is a fraction of that value. Then Algorithm 2 may take a very long time to converge, trying to reach $\max(f)$, in what can be considered a waste of computational effort. Because if there are rare states with much less importance than $\max(f)$, most observations of the rare event will involve these states, and not those realizing the maximum value of f . This issue is further discussed in Sections 3.5 and 3.6.

3.3.4 Estimation and convergence

Together with Algorithm 1 to derive the automatic I-FUN, and letting momentarily aside the particular splitting technique to use, Sections 3.3.1 to 3.3.3 supply enough mechanisms to implement automated multilevel splitting *simulations*. A simulation is the execution of any I-SPLIT technique as introduced

in Section 2.5.2, e.g. a Fixed Effort run to study some transient property, or a long run of RESTART in batch means to analyse steady-state behaviour.

Each simulation yields a point estimate $\hat{\gamma}_i$ for the probability γ of the rare event. The next step is using the statistical theory from Section 2.3.3 to analyse the sample $\{\hat{\gamma}_i\}_{i=1}^N$ and produce the desired interval estimate around the mean of the data, $\hat{\gamma} \doteq 1/N \sum_{i=1}^N \hat{\gamma}_i$. The intention is to provide the user with a reliable guess of γ , where reliability is quantified in terms of confidence coefficient and interval precision.

The confidence interval from Definition 5 and Equation (6) in Section 2.3.3 assumes the population mean is estimated without information about the distribution of the samples. Since the variance σ^2 is unknown and approximated with the estimator S_N^2 for a sample of size N , the Central Limit Theorem is used with the Student's t-distribution to guarantee

$$P\left(\mu \in \left[\bar{X} \pm t_{\frac{\alpha}{2}} \sqrt{\frac{S_N^2}{N}}\right]\right) \approx 1 - \alpha \quad (12)$$

where t_α is the α -quantile of the Student's t-distribution, $\bar{X} = \hat{\gamma}$ is the mean value of the random sample $\{X_i\}_{i=1}^N = \{\hat{\gamma}_i\}_{i=1}^N$, and $\mu = \gamma$ is the unknown population mean.

However, when a transient analysis is performed, each path will either find a rare state or get prematurely truncated upon encountering a stopping state. Thus each simulation can be regarded as a Bernoulli trial, where observing the rare event means success.

In such setting, running N simulations is equivalent to performing an $\langle N, \gamma \rangle$ -Binomial experiment, where the variance of each Bernoulli trial is characterised by the expression $\sigma^2 = \gamma(1 - \gamma)$. Using $S_N^2 = \hat{\gamma}(1 - \hat{\gamma})$ to estimate the variance of the population, eq. (12) then yields the CI

$$\hat{\gamma} \pm t_{\frac{\alpha}{2}} \sqrt{\frac{\hat{\gamma}(1 - \hat{\gamma})}{N}}. \quad (13)$$

For some confidence criteria provided by the user, i.e. confidence coefficient α and interval width d , this suggests the following approach to generate the CI: increasingly generate samples, viz. simulation paths, computing each time the value of $t_{\frac{\alpha}{2}} \sqrt{\hat{\gamma}(1 - \hat{\gamma})/N}$; as soon as it falls below $d/2$, the desired criteria is satisfied and estimation finishes.

Recall however that the model and property queries considered are in a rare event regime, where asking for relative error is more robust than

requesting some width d fixed a priori. Moreover the expression of eq. (13) is unreliable for the extreme cases when $\gamma \approx 0$ and $\gamma \approx 1$. There are more robust estimators, like the Wilson score interval [Wil27], which cover such cases with better accuracy.

With those considerations in mind, this approach based on the transient nature of the simulations provides a simple convergence decision mechanism. Unfortunately, its apparent suitability notwithstanding, it cannot be dependably used in general as stopping criterion.

The problem stems from the use of multilevel splitting. When simulations are standard Monte Carlo, each sample has a success/failure outcome. Instead when using e.g. RESTART with stacked splitting factor K , the result can take any value in $\{0, 1/K, 2/K, \dots, K-1/K, 1\}$ or even be unbounded—see Section 2.6. Therefore, and in general for any splitting technique, the random sample $\{X_i\}_{i=1}^N$ does not necessarily follow a $\langle N, \gamma \rangle$ -Binomial distribution[†].

For the estimation approach followed in this work, the complication described above materialises in a premature stop: the convergence criterion deems the current data set sufficient, when in truth more simulations were needed to build an interval containing γ with the desired confidence.

The problem of real parameter coverage is one of the most difficult to solve in RES [GRT09]. One strategy is to use the standard CI expression of eq. (12), based on the Central Limit Theorem which makes few assumptions on the random sample. Notice anyway that in a typical scenario where the distribution of the simulated paths is unknown, the sample size required to satisfy the confidence criteria can only be discovered a posteriori. This can be dangerous when the simulation budget is fixed.

This analysis has certain correspondence with the use of a relative error to define the desired interval width, since that also implies simulating first and estimating later, with no foreseeable notion of termination. Reijsbergen et al. study analogous complications in [RdBS16], when importance sampling is used for hypothesis testing.

Prioritising dependability over speed of convergence, this thesis uses the expression from eq. (12) to build the CI in both transient and steady-state analysis. Replacing the standard statistical nomenclature with their RES counterparts, the resulting equation applied for estimations is

$$P\left(\gamma \in \left[\hat{\gamma} \pm t_{\frac{\alpha}{2}} \sqrt{\frac{\hat{\sigma}_{\hat{\gamma}}}{N}}\right]\right) \approx 1 - \alpha \quad (14)$$

[†] Similar issues are known to affect importance sampling [RdBS16].

where $\hat{\sigma}_{\hat{\gamma}}$ is the empirical variance of the sample $\{\hat{\gamma}_i\}_{i=1}^N$ (see eq. (4)), and $t_{\frac{\alpha}{2}}$ is the $\alpha/2$ quantile of the Student's t-distribution.

The conventional lower bound $N \geq 30$ is imposed, after which comparisons for convergence start. Each new sample (viz. simulation result) updates the value of the estimate $\hat{\gamma}$ and thus also of the desired interval width, by means of the relative error approach. Each sample updates also the empirical width of the computed CI following eq. (14). When this empirical width becomes narrower than one derived from $\hat{\gamma}$ as relative error, convergence is assumed and the resulting CI is returned.

Remarkably, this estimation strategy is equivalent to the ‘‘Chow-Robbins test’’ from [RdBS16], reported in that work as the only alternative which can yield a correct estimate regardless of the true sample distribution. The price to pay is not being able to foretell how many simulation paths are needed to satisfy the user’s confidence criteria.

3.4 Tool support

We developed the software tool BLUEMOON, implementing the automatic approach to importance splitting described in the previous section. It was written in C++ and Java as a modular extension of the probabilistic model checker PRISM [KNP11], development version 4.3, which runs on the Java Virtual Machine.

The BLUEMOON tool is free and open software, released under the terms of the General Public License (GPL v3). The source code can be downloaded from the homepage of the tool, located in the webpage of the Dependable Systems Group at <http://dsg.famaf.unc.edu.ar/tools>.

It is important to highlight the prototypical nature of BLUEMOON, which was devised to validate the theory presented in this chapter. The desire for empirical validation motivated the choice of continuous and discrete time Markov chains. Many studies already exist for this kind of systems, which facilitated the task of reproducing known results.

Notwithstanding the above, and as discussed in Sections 3.2.3 and 3.3.3, the algorithms and techniques introduced do not make any assumption of memorylessness. They can thus be employed to study more general stochastic processes, as we will show in Chapter 4.

As an extension of the PRISM tool, BLUEMOON reads DTMC and CTMC models described in the PRISM input language, and property queries expressed in the PRISM property specification language—see Sections 3.3.1 and 3.3.2.

The model and queries are written down in a text file and the tool is invoked in the command line, passing as input the file and the options `--rarevent <type> <strategy>` and `--rareconf <conf> <prec>`. Arguments are mandatory; their syntax and semantics is as follows:

- `<type>` Specifies whether the analysis is transient or steady-state, which is respectively indicated with the values `tr` and `ss`.
- `<strategy>` Specifies which kind of simulations shall be run. Its value must be one of the following:
 - `nosplit` to use the standard Monte Carlo approach;
 - `auto` to use the importance splitting approach with the automatic importance function, derived from the user model and query using Algorithm 1;
 - `adhoc` to use importance splitting but with an ad hoc importance function, which requires the user to define it.
- `<conf>` Specifies the confidence coefficient desired by the user and must thus be a (rational) number in the open interval $(0, 1)$.
- `<prec>` Specifies the interval precision, and can be either a fixed rational number, or a percentage with the format `p%` to use the relative error approach, where $p \in \{1, 2, \dots, 100\}$ is interpreted on the full width of the interval.

For instance the line

```
>_ prism-bm model.prism --rarevent tr auto --rareconf .9 20%
```

invokes the tool (identified by the command `prism-bm`) on the PRISM model file `model.prism`, to run a transient analysis using importance splitting with the automatic I-FUN derived by BLUEMOON, requesting a confidence level of 90% and a relative error of 10%—the empirical width of the CI must be at most 20% of the estimate, i.e. smaller than $0.2 \times \hat{\gamma}$ —see Definition 6.

There is also a `--rareparams` option which takes as argument a comma-separated list of customizations, like the splitting to use, the confidence interval building strategy, a wall-clock execution timeout, etc. One of its most relevant uses is to define the importance function when the ad hoc approach is selected. The function must be an integer-valued arithmetic expression on the variables and constants of the model. For instance the command

```
>_ prism-bm model.prism --rarevent ss adhoc \
    --rareconf .95 20% \
    --rareparams "timeout=5,ifun=acc^2-5*q2"
```

runs a steady-state analysis of `model.prism`, using I-SPLIT with the ad hoc importance function which subtracts five times the value of `q2` from the square of `acc`. Both `acc` and `q2` should be defined in `model.prism`, either as constants or variables, and the expression must evaluate to an integer.

The ad hoc importance function can also be defined as a PRISM formula inside of the model. The example above is equivalent to appending “`formula importance = acc^2-5*q2`” as new line to the `model.prism` file, and then executing the same command but with `timeout=5` as only element of the list passed as argument to the `--rareparams` option.

When the automatic I-FUN construction is selected, Algorithm 1 is used to build an explicit function on the state space of the global system model. This means the importance value of each concrete state is stored as an integer in a vector. Interestingly, the backwards BFS of the algorithm uses a column major sparse matrix representation (CSC) of the adjacency graph, which eases the reversed traversing of the model transitions. Since simulations need to take transitions in a forward manner, the matrix is made row major (CSR) once the importance function has been built.

The multilevel splitting technique implemented in the BLUEMOON tool is RESTART. Whenever the `auto` or `adhoc` argument is passed to the option `--rarevent`, the thresholds are selected for the corresponding I-FUN using Algorithm 2. Once the thresholds are ready, RESTART simulations are executed to generate the collection $\{\hat{\gamma}_i\}$ of estimates, from which the CI is built as detailed in Section 3.3.4.

A global splitting value is used for all thresholds. By default it equals two, meaning each time a trial crosses a threshold upwards, two trials will continue execution, i.e. one clone is created. This value can be tuned with the `split=<num>` customization of the `--rareparams` option.

The global splitting value influences the selection of the thresholds by means of the *balanced growth* approach [Gar00, eq. (2.25)]. Generally speaking, the higher the splitting value, the further the thresholds will be from each other. The aim is to have roughly the same level-up probability (Definition 10) in all importance levels, since that should increase the efficiency of RESTART [VAMGF94].

The main output of the tool displays the resulting point estimate $\hat{\gamma}$, the precision of the CI, and the interval itself. It also shows the current stage of

the execution: building the model, the importance function, the thresholds, etc. Specifically when simulating, it shows how many samples $\hat{\gamma}_i$ have been generated so far. An extract of an output is

```
>_ PRISM
=====

Version:  4.3.dev
      :
Type:     CTMC
Modules:  ContinuousTandemQueue
Variables: q1 q2 arr lost

-----
[DEV] Rare event simulation chosen.
[DEV] Simulation type: TRANSIENT
[DEV] Simulation strategy: RESTART_AUTO
      :
Identifying special states... done.
Building importance function... done.
Setting up RESTART simulation environment... done.
Estimating rare event probability {5}{6}{8}{52} done:
- Point estimate: 5.873E-6
- Precision: 1.175E-6
- Confidence interval: [ 5.286E-6 , 6.46E-6 ]
      :
```

When estimations finish successfully, like in the example above, some timing information is printed after the numerical estimates. The total wall-clock execution time is discriminated in the different stages composing the full execution. A sample output (continuation of the above) is:

```
>_
      :
- Confidence interval: [ 5.286E-6 , 6.46E-6 ]

Processing times information
- Total elapsed time: 29.18 s
- Setup time: 0.68 s
  > Initial setup: 0.02 s
```

```

> Rare/Reference states identification: 0.01 s
> Importance function building: 0 s
> Thresholds selection: 0.65 s
- Simulation time: 28.5 s

[DEV] Skipping other model checks.

```

However execution can be prematurely interrupted, truncating estimations before the desired confidence criteria has been met. This can happen either by reaching a predefined timeout (set with the `timeout=<num>` customization of `--rareparams`), or by a user or system interrupt. If simulations had already started and there is estimation data available when interrupted, BLUEMOON shows the point estimate reached plus CI for typical confidence levels. A sample output is:

```

>_
:
Estimating rare event probability {8}{12}{13}{41} wall time
limit reached.

[rareevent.RareeventSimulatorEngine] Interrupted, shutting
down
- Point estimate: 4.851E-6
- 90% confidence: precision = 1.285E-6
                   interval = [ 4.208E-6 , 5.493E-6 ]
- 95% confidence: precision = 1.531E-6
                   interval = [ 4.085E-6 , 5.616E-6 ]
- 99% confidence: precision = 2.012E-6
                   interval = [ 3.845E-6 , 5.857E-6 ]

```

Besides its main output, BLUEMOON has a technical output where execution steps are described in more detail. Information like which concrete states are rare/stopping, the seed fed to the random number generator, the importance value of the thresholds, and even an extract of the importance function, are printed in the technical output of the tool.

This data is useful for analysing an execution in depth, e.g. when we wish to compare the thresholds selected, or for debugging. By default it is dumped as plain text in a file named after the execution, so for instance

```

>_ prism-bm model.prism --rareevent ss auto ...

```

will print technical info into “`rarevent_STEADYSTATE_RESTART_AUTO.log`.” This can be changed with `techlog=<name>` in the `--rareparams` option.

The features offered by BLUEMOON can be queried via the PRISM help interface. For example the command `prism-bm --help rareparams` displays all the customizations the `--rareparams` options has to offer. Interested readers are referred to the tutorial in the homepage of the tool, where some use cases are illustrated with the tandem queue model.

3.5 Case studies

Several examples were taken from the RES literature and analysed with BLUEMOON. The general description of the systems and the results from experimentation are shown here. The models used to generate this data are listed in Appendix A.

3.5.1 Experimentation setting

All models studied are continuous or discrete time Markov chains described in the PRISM input language. Consequently, we could use the model checker to validate that the models implemented produce the desired outcomes, i.e. the values published in the works we took them from.

We launched independent experiments for each case, estimating intervals for confidence coefficients and relative errors fixed a priori. All experiments ran until the confidence criteria were met, or a wall-clock execution time limit (*wall time limit*) was reached. The hardware used was a 12-cores 2.40GHz Intel Xeon E5-2620v3 processor, with 128 GiB 2133MHz of available DDR4 RAM. We point out however that BLUEMOON uses one core per estimation.

For each system we varied some parameter, stressing out the convergence conditions by increasing the rarity of the event hence decreasing the value of γ . For each model and parameter value, we tested three simulation strategies: RESTART using the automatic I-FUN, RESTART using ad hoc importance functions (some taken from the literature), and standard Monte Carlo. Four global splitting values were tested in the I-SPLIT simulations.

We checked the consistency of the confidence intervals obtained, comparing them against the values produced by the PRISM tool. This section presents charts displaying the convergence time for each strategy. Time measurements cover the full computation process, including preprocessings like the compilation of the model file and the selection of the thresholds. We

repeated each experiment thrice; the values shown are the average of the wall execution times measured for each experiment.

We present all results displaying one chart per splitting value tested. The outcomes of the standard Monte Carlo simulation appear repeated in all charts to ease visual comparisons. In each case, we span along the x-axis the parameter varied to increase the rarity of the event. On the y-axis we show the average time to convergence, in seconds and using a logarithmic scale.

On the one hand, a bar reaching the upper border of the chart signifies timeout prior to convergence, and is denoted a *failure*. On the other hand and unless noted otherwise, all simulations which converged before the wall time limit produced an interval containing the value computed by PRISM for the exact same model.

3.5.2 Tandem queue

Transient analysis

Recall the tandem Jackson network presented in Example 2 consisting of two connected queues. For a continuous time setting, we replicate the experiment of [Gar00, p. 84], using parameter values $(\lambda, \mu_1, \mu_2) = (3, 2, 6)$. Starting from state $(q_1, q_2) = (0, 1)$, we are thus interested in observing a second queue fully occupied (denoted a *saturation* in the second queue) before it empties. The property query we used is $P=? [q_2 > 0 \cup q_2 = c]$, where variable c is the maximum queue capacity (C).

We used the PRISM model from Appendix A.1. Notice we represent the queue monolithically, in contrast to the modular implementation previously shown in Code 3.3. Both models are semantically equivalent, but the monolithic version makes it easier to signal events, like an external packet arrival, without the need to use global variables.

Notice also that the service rate at the second queue, μ_2 , is greater than the one at the first queue. That means the first queue is the bottleneck and hence the rarity of the saturation comes from the fast service times at the second queue. According to [VAVA06, VA07b, LLGLT09] in respect of the tandem queue, this is the most difficult scenario to solve.

We tested maximum capacities $C \in \{8, 10, 12, 14\}$, for which the values of γ approximated by PRISM are respectively 5.62e-6, 3.14e-7, 1.86e-8, and 1.14e-9. A 95|10 CI criterion was imposed. This means estimations had to reach a 95% confidence level and 10% relative error, i.e. the empirical precision of the interval had to be smaller than 0.2 times the estimate $\hat{\gamma}$. This was to be achieved within 3 hours of wall time execution.

For the importance splitting simulations, besides the automatic I-FUN computed by BLUEMOON (denoted `auto`), we tested three ad hoc importance functions: counting the number of packets in the second queue alone (q_2), counting the packets in both queues (q_1+q_2), and a weighed variant of that second function (q_1+2*q_2). We used the global splitting values 2, 5, 10, and 15. Standard Monte Carlo simulations are denoted `nosplit` in the charts and throughout this section.

The average wall execution times to convergence are shown in Figure 3.3. Recall we display one chart per splitting value, with the outcomes of the `nosplit` simulations repeated in all four charts. Moreover, since the parameter varied to increase the rarity of the event is the maximum queue capacity, we span the tested values of C along the x-axis.

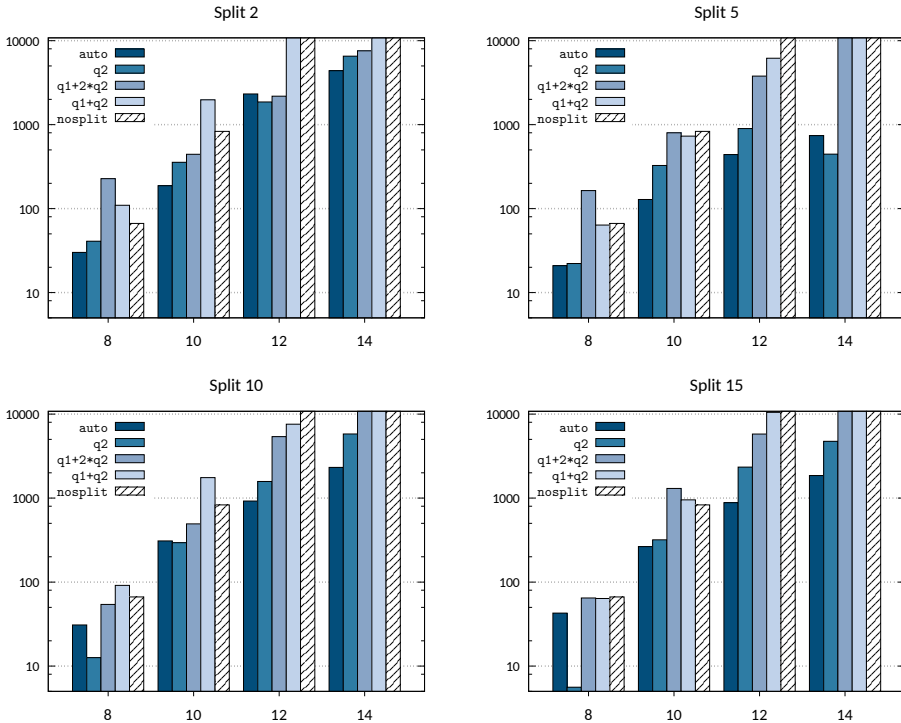


Figure 3.3: Transient analysis times of tandem queue (CTMC model)

For the higher values of C and as expected, the standard Monte Carlo simulations failed, i.e. they could not meet the the criterion chosen for the confidence intervals within the time limit imposed.

Outstandingly and with few exceptions (e.g. for $C = 8$ with split 10 and 15), the auto importance function outperformed all ad hoc variants in most configurations. The closest competitor was `q2`, which sometimes resembled or improved the convergence times of `auto`, most notably for the smallest queue size (where the event is not so rare).

In general, results seem to indicate that the global splitting strategy implemented in BLUEMOON is quite sensitive to the value chosen. In particular, Figure 3.3 suggests 5 is the best option among the four splittings values used for experimentation. In that respect the auto importance function showed less variance than `q2`; compare e.g. the performance of these two functions for the different splitting values when $C \in \{8, 14\}$.

As explained in Section 3.4, BLUEMOON employs the balanced growth approach when automatically selecting the thresholds, in an attempt to reduce the variability of RESTART due to the relationship between the thresholds location and their splitting. The apparently unpredictable behaviour observed when varying the splitting value seems to indicate that the chosen strategy is suboptimal. Further discussions on this topic can be found in the following sections.

Last, the convergence times of the ad hoc variants showing the worst performance are noteworthy. For some splitting values when $C \in \{8, 10\}$, both `q1+q2` and `q1+2*q2` took longer even than `nosplit` simulations. This is evidence that without a proper choice of importance function, thresholds, and splitting values, the computation overhead of splitting techniques like RESTART can degrade performance.

Steady-state analysis

We also studied the steady-state behaviour for the saturation of the second queue. Here γ stands for the time proportion the second queue spends in a saturated state during long runs. The corresponding query is `S=? [q2=c]`.

For the maximum capacities tested $C \in \{10, 15, 20, 25\}$, the values of γ approximated by PRISM are 3.36e-6, 1.62e-8, 7.42e-11, and 3.29e-13. Estimations had to build a 95% CI within 2 hours of wall time execution. We tested the same importance functions and splitting values as in the transient analysis. Results are presented in Figure 3.4.

Standard Monte Carlo simulations converged only for the smallest queue capacity. This was expected since, except for $C = 10$, the queue capacities used in this experiment exceed the ones of the transient analysis, where `nosplit` simulations had failed for the highest values of C .

Prominently, all standard Monte Carlo experiments either failed, or

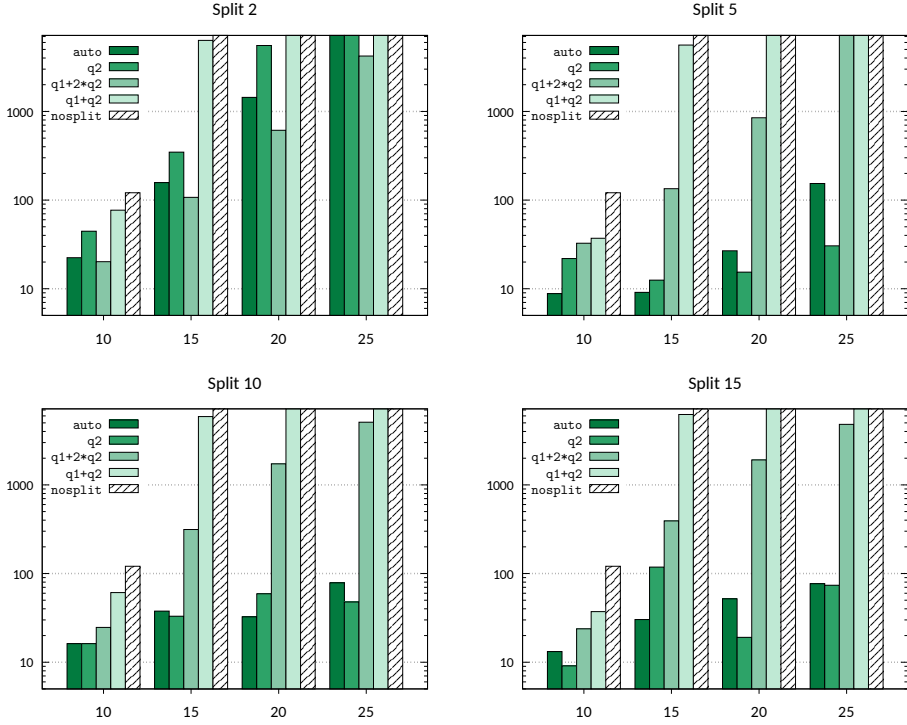


Figure 3.4: Steady-state analysis times of tandem queue (CTMC model)

took longer to converge than the runs using importance splitting, whichever splitting value was selected. This suggests RESTART could be better fitted to perform steady-state rather than transient analysis, at least in the sequentially connected queueing setting of this tandem queue.

Again the auto importance function was either the best or the runner up in terms of performance. From the four splitting values tested, it converged the slowest for split 2. Its general behaviour did not vary much among the other splitting values, unlike its closer competitor, namely q_2 .

It is noteworthy that in a few cases, convergence times decreased as the rarity of the event increased. See e.g. q_2 with splitting 5 for queue capacities $C \in \{10, 15, 20\}$, and also q_2 with splitting 15 for queue capacities $C \in \{15, 20, 25\}$. Studying the technical output of BLUEMOON reveals the reason may be the automatic selection of thresholds. We base this conjecture on the reasons exposed next.

Take for instance the performance of q_2 for splitting 15, where BLUEMOON selected 5–6 thresholds for $C = 15$, 8–9 thresholds for $C = 20$, and 8–11

thresholds for $C = 25$. Since the number of thresholds is almost the same for $C \in \{20, 25\}$, the convergence times are mostly influenced by the rarity of the event, viz. the size of the queue. This results in longer convergence times for $C = 25$ than for $C = 20$, as Figure 3.4 shows (and as expected).

For $C = 15$ however, convergence took longer than for $C \in \{20, 25\}$. Furthermore, measurements are consistent in the three experiments we ran for this configuration, which used different seeds of the random number generator. This is clearly at odds with the expected behaviour.

The most plausible explanation seems to be the number of thresholds: the algorithm chose too few of them for $C = 15$, hence the full gain derived from the use of splitting could not be achieved, and the performance of the I-SPLIT simulations was even worse than for $C = 25$.

Such theory is also supported by the experiments which did behave as expected, that is, where convergence times increased together with the value of C . The conjecture is that in these cases, an increment in the value of C should be reflected in an increment in the number of thresholds, particularly with (strictly) more than six thresholds for $C = 15$.

Take for example the case of the auto importance function for splitting 2. The number of thresholds automatically selected for $C = 10, 15, 20, 25$ was respectively 3, 8, 13, and 18, and these numbers were consistent in all (three) experiments run for each configuration. Something analogous is observed for splitting 10 with the importance function q_1+2*q_2 , where the number of thresholds automatically selected were 2-3, 7, 9-10, and 13, respectively for the values of $C = 10, 15, 20, 25$. This is thus more evidence in favour of the conjecture that the unexpected higher times for smaller values of C could be caused by a bad selection of thresholds.

These kind of anomalies suggest an inefficient implementation of RESTART, derived from a suboptimal thresholds selection mechanism. The situation is similar (and assumed related) to the variability in performance due to the splitting value used, which was observed in the previous transient study. We discuss possible solutions to this problem in Section 3.5.3.

3.5.3 Discrete time tandem queue

We also studied the tandem queue in a discrete time setting. Recall the single queue presented in Example 4 of Section 3.3.1. Here as well *time ticks* mark the discrete evolution points of the system, in a scenario where multiple events can take place at the same tick (e.g. an external packet arrival and a packet service in the second queue).

As before, interest lies in studying a system where the first queue is the bottleneck. The rarity of the saturated state in the second queue, $q2=c$, is due to its fast service times w.r.t. the first queue.

Since the model is discrete, the rates from the continuous scenario need to be replaced with probabilities, defining the odds of an event taking place at each time tick. Let thus `parr` denote the probability per time tick of an external packet arrival, and `ps1` and `ps2` the probability per time tick of a packet service in the first and second queues respectively. We used the PRISM model from Appendix A.2. Notice that just like in the continuous case, we implemented the queue as a single module rather than compositionally[†].

We carried out a steady-state analysis for probabilities `parr` = 0.1, `ps1` = 0.14, and `ps2` = 0.19 and maximum capacities $C \in \{10, 15, 20, 25\}$. The corresponding long run saturation values (γ) approximated by PRISM are 4.94e-7, 1.28e-8, 3.22e-10, and 7.96e-12. Estimations had to build a 90\10 CI within 4 hours of wall time execution.

Importance functions similar to those tested in the continuous case were used in the I-SPLIT simulations. Namely besides `auto`, the ad hoc functions employed were `q2`, `q1+q2`, and `q1+5*q2`. We tested the global splitting values 2, 5, 10, and 15. Figure 3.5 displays the average wall times measured.

Unlike with the continuous time model, the smallest splitting value tested yielded the shortest convergence times. Remarkably, the `auto` importance function was the fastest to finish in that setting, for $C \in \{15, 25, 30\}$. Moreover, leaving aside the splitting value 5, an excellent performance of `auto` is observed. The second best I-FUN is clearly `q2`, just like in the experiments with the CTMC model of the tandem queue.

The resulting wall execution times for splitting value 5 deserve special attention. For the values $C \in \{15, 20, 25\}$, `auto` took considerably (and inconsistently) longer than `q2`. We studied each individual experiment in further depth, and draw the following conclusions.

In the case of $C = 15$, two experiments of `auto` took less than 5 seconds and one took 16 seconds, whereas all experiments of `q2` took around 5 seconds. Since the number of thresholds did not vary much we attribute this to a bad seed of the random number generator. The influence of such incident is exacerbated by the small computation times, resulting in a high relative variance. The same experiments were repeated with different seeds of the random number generator, and the convergence times observed were very

[†] For a DTMC in the PRISM language this implies stating all possible events at each time tick; that is why the discrete model almost triples in length its continuous counterpart.

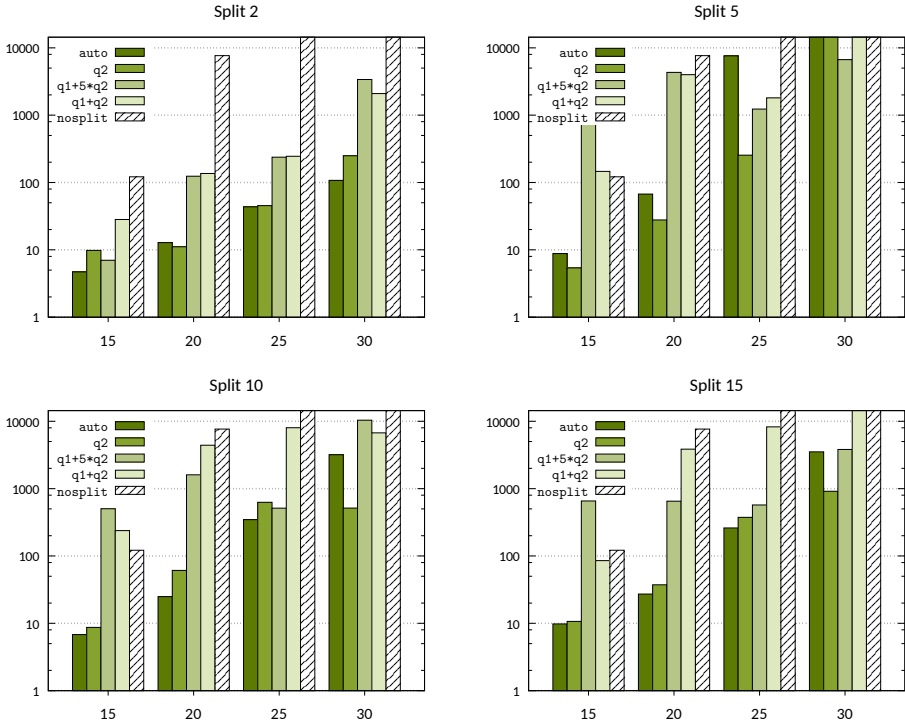


Figure 3.5: Steady-state analysis times of tandem queue (DTMC model)

similar between auto and q_2 , thus supporting this explanation.

For $C \in \{20, 25\}$ the situation is quite different. For these cases we observed that *fewer thresholds* tend to yield *faster convergence times*, contrary to the overall observations in the continuous time case. We also witnessed this behaviour in the outcomes of experimentation with a splitting value of 10, but not in the experiments with a splitting of 2.

A possible explanation is that the theory used for the selection of the thresholds, which is based on a *global* (unique) splitting value, is inadequate for purely discrete systems. Already for the CTMC model of the tandem queue, there is evidence that the implementation in BLUEMOON of the thresholds selection mechanism is not optimal. In a DTMC model not only the state space but also the transitions of the system are discretised in time. In this setting, tiny variations in the splitting or the thresholds may have a snowball effect in RESTART, causing starvation or overhead in the upper importance levels hence degrading performance.

One solution to this problem could be to increase the effort spent in selecting and probing the thresholds. In that respect, Algorithm 2 implements the technique Adaptive Multilevel Splitting, for which an improved version is known. This newer version is named Sequential Monte Carlo [CDMFG12], and it enhances the statistical properties of the algorithm, reducing the variance of the outcome. An alternative way to tackle with the issue would be to reduce the snowball effect derived from having a single global splitting value, choosing instead the splitting for each threshold individually.

A last minor remark: the average execution times of q_1+5*q_2 for split 5 and $C = 15$ is concealed by the legend of the chart. In that configuration the I-FUN took 6535.5 s to converge. This is almost as much as it took for the same splitting but $C = 30$, where it converged in 6674.7 s.

3.5.4 Mixed open/closed queue

Consider a queueing network consisting of two *parallel* queues handled by a single server. An *open queue*, q_o , receives packets at rate λ from an external source. A *closed queue*, q_c , receives (sends) packets from (to) an internal system buffer. The same server processes packets in both queues, giving priority to the closed one. That is, packets in q_o are served at rate $\mu_{1,1}$, unless q_c has packets, which will be served first at rate $\mu_{1,2}$. In turn, packets in *internal circulation* are processed in the system buffer at rate μ_2 , and sent back to q_c . When a single packet is in internal circulation, the network is actually an $M/M/1$ queue with server breakdowns. A schematic representation of this system is show in Figure 3.6.

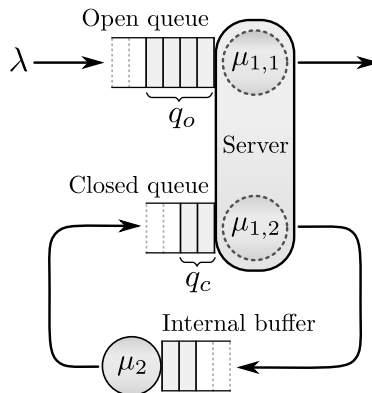


Figure 3.6: Mixed open/closed queue

This was studied in [GHSZ99, Sec. 4.1] in a continuous time setting. Starting from an empty state $(q_o, q_c) = (0, 0)$, a transient analysis was performed to estimate the probability of q_o reaching some maximum capacity B , before both queues become empty again. The setting studied in that work has a single packet in internal circulation, rates $\lambda = 1.0$, $\mu_{1,1} = 4$, $\mu_{1,2} = 2$, $\mu_2 \in \{0.5, 1.0\}$, and capacities $B \in \{20, 40\}$.

We used the PRISM model from Appendix A.3. The implementation is modular, although the variables representing queue occupancy have global scope. The transient probability query was `P=? [!reset U lost]`.

We analysed the transient behaviour of this system in the setting of [GHSZ99], for maximum capacities $B \in \{20, 30, 40, 50\}$ of q_o . The corresponding probabilities of the rare event approximated by PRISM for rate $\mu_2 = 1.0$ are 5.96e-7, 5.82e-10, 5.68e-13, and 5.55e-16. Instead for $\mu_2 = 0.5$ they are respectively 3.91e-8, 8.89e-12, 2.02e-16, and 4.60e-19. Estimations were set to build a 95\|10 CI within 2.5 hours of wall time execution.

RESTART simulations featured the `auto` importance function and three ad hoc variants: counting solely the packets in the open queue (`oq`), adding information of whether the packet in internal circulation is currently in q_c (`cq+oq`), and a weighed version of that last variant (`cq+5*oq`). We ran experiments for the global splitting values 2, 5, 10, and 15.

The resulting execution times for an internal server with rate $\mu_2 = 1.0$ are presented in Figure 3.7. Standard Monte Carlo simulation failed for all but the smallest queue size, $B = 20$. This comes as no surprise given the values of γ approximated by PRISM for $B \in \{30, 40, 50\}$. Still, these results provide further evidence that (a reasonable implementation of) importance splitting is a good choice when analysing the properties of a model in a rare event regime. Impressively, almost all I-SPLIT simulations took less than a minute to converge, yielding confidence intervals which contained the values of γ approximated by PRISM. Instead, the only `nosplit` simulations to converge took nearly 40 minutes, and this in the less demanding setting.

Studying the performance of the different importance functions, this is the first situation where `auto` is not among the favourites. It behaved quite well for $B \in \{30, 40\}$ with splitting 2, which was the splitting yielding the best performance in most configurations. In general however it was slower to converge than the ad hoc variants, though together with `cq+oq` it showed smaller sensitivity to the splitting value than the other two functions.

This is also the first situation where the more complex ad hoc importance functions rivalled the plain count of packets in the target queue, i.e. `q2`. The best average convergence times where: 10.2 seconds for $B = 20$, achieved by

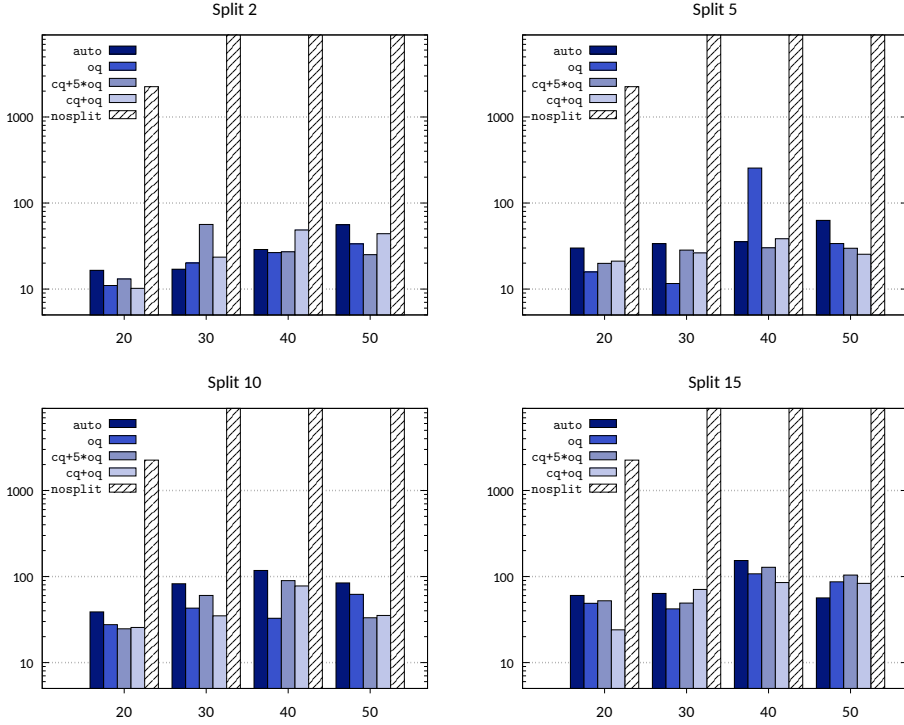


Figure 3.7: Transient analysis times of mixed open/closed queue ($\mu_2 = 1.0$)

cq+oq with splitting 2; 11.6 seconds for $B = 30$, achieved by oq with splitting 5; 26.5 seconds for $B = 40$, achieved by oq with splitting 2; and 25.1 seconds for $B = 50$, achieved by cq+5*oq with splitting 2.

The incongruously long average convergence time of oq for $B = 40$ with splitting 5 deserves special attention. When we looked at the individual experiments the cause became evident: the third experiment took 12 minutes, whereas the other two converged in 33 and 12 seconds. Once again, the anomaly can be explained by looking at the thresholds. The outlier used 18 of them, and the other two experiments used 13 and 14 thresholds respectively.

This case (oq with splitting value 5 and $B = 40$) is a good example of the starvation/overhead dichotomy, which affects RESTART when the thresholds and the splitting are not selected properly. With 13 thresholds estimations converged in 33 seconds. In the same setting but with 14 thresholds things improved, needing only 12 seconds to meet the confidence criteria. This suggests that 13 thresholds yield too little splitting, causing some simulations

to starve and not reach the rare event. But then with 18 thresholds there is too much splitting and truncation going on, derived from an overhead in the number of simulations. Looking at the results of each batch in the technical output, it is clear that the variability of the outcomes is too high, and thus statistical convergence takes longer.

Figure 3.8 shows execution times for the alternative rate $\mu_2 = 0.5$. The outcomes of experimentation with $B = 50$ are omitted because all of them failed to converge within the wall time execution limit of 2.5 hours. Recall PRISM computed a probability value of $4.6e-19$ to observe such transient event. Since that is three orders of magnitude lower than the case of $B = 50$ for $\mu_2 = 1.0$, these failures are not particularly surprising. Neither is the fact that all standard Monte Carlo simulations failed as well.

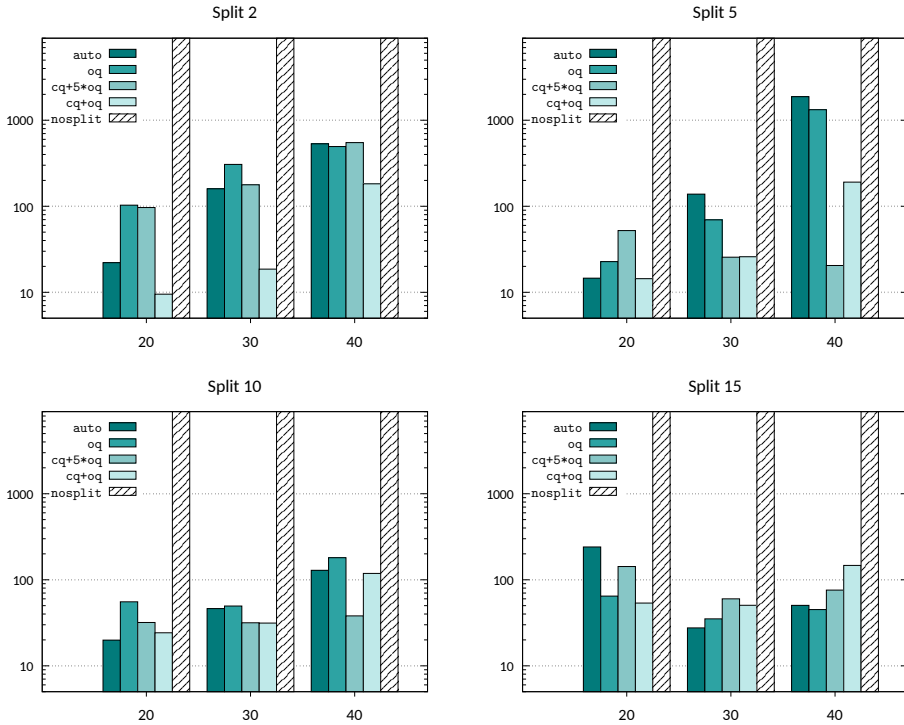


Figure 3.8: Transient analysis times of mixed open/closed queue ($\mu_2 = 0.5$)

To interpret the results shown in Figure 3.8, it is crucial to understand what does the change in the value of μ_2 imply. Recall μ_2 is the rate at which a packet is sent from the internal buffer to the closed queue (q_c). Furthermore,

recall that the server attending the open queue (q_o) will not dequeue packets from q_o as long as q_c is not empty. This was analogous to having a broken server attending q_o . Therefore, a lower value of μ_2 implies *longer periods* of the server *dequeueing packets from q_o* , resulting in lower probabilities of observing the rare event of a saturated open queue.

As μ_2 decreases, having a packet in q_c (viz. a broken server) becomes less common, and the state of q_c *grows in importance*. That is likely the reason why, for most splitting values, the importance functions $cq+oq$ and $cq+5*oq$ converged faster than oq . Contrasting against the previous setting where $\mu_2 = 1.0$, the performance of the two functions which consider q_c improved w.r.t. oq , here where $\mu_2 = 0.5$.

Doing without these kind of analyses is precisely one of the goals behind the automatic derivation of the importance function. Notice that the `auto` I-FUN converged faster than oq in many situations, and even better than the best candidates ($cq+oq$ and $cq+5*oq$) in a few cases, e.g. $B \in \{30, 40\}$ with splitting 15 and $B = 20$ with splittings 5 and 10. That is why, even though `auto` was not the best performing importance function, we still consider these results to be satisfactory.

The high variability among the different splitting values tested deserves the same considerations as in the previous case studies. Finally, we observe that the slower convergence of `auto` with splitting 15 for $B = 20$ w.r.t. $B \in \{30, 40\}$, was caused by one of the three experiments. Two runs converged in less than 2 minutes, whereas the third experiment took 10 minutes. This problem, already observed for the DTMC representation of the tandem queue, could be mitigated by increasing the number of experiments repeated for each configuration.

3.5.5 Queueing system with breakdowns

Recall the system presented in Example 1, where several sources (which can be of either one of two types) send packets to a single buffer, and all of them can become non-operational and get repaired afterwards. The single server attending the buffer also breaks down and gets repaired, so this system can be regarded as a generalisation of the mixed open/closed queue from the previous section.

Kroese et al. studied such a process in [KN99], using importance sampling in a continuous time setting with five sources of type 1 and also another five of type 2. The rates used in [KN99, Sec. 4.4] are: $(\alpha_1, \beta_1, \lambda_1) = (3, 2, 3)$ for sources of type 1, describing the speeds of repair, failure, and packet

production respectively; $(\alpha_2, \beta_2, \lambda_2) = (1, 4, 6)$ for sources of type 2; and rates $(\delta, \xi, \mu) = (3, 4, 100)$ for the server, where μ stands for packet processing rather than packet production.

We used the PRISM model from Appendix A.4. The implementation is monolithic, taking advantage of some Markovian properties to reduce the size of the state space[†]. Starting with a single packet enqueued in the buffer, a broken server, and a single operational source of type 2, the rare event is a buffer saturation for some maximum capacity K before it empties. The corresponding transient property is $P=? [\text{buf}>0 \text{ U } \text{buf}=\mathbf{k}]$, where \mathbf{k} is K , the maximum buffer capacity to reach.

We performed a transient analysis for maximum capacities $K \in \{40, 80, 120, 160, 200\}$. The corresponding values of γ obtained with PRISM are 4.59e-4, 3.72e-7, 3.02e-10, 2.45e-13, and 1.98e-16. We used a 95\10 CI criterion together with a wall time execution limit of 3 hours. Like in the previous case studies, the splitting values tested in the importance splitting simulations were 2, 5, 10, and 15.

Besides the `auto` I-FUN computed by BLUEMOON, we tested three ad hoc importance functions. The simplest one, `buf`, just counts the number of packets in the buffer. That seems too naïve, since the up/down state of the sources is crucial to generate the desired saturation. Therefore, another variant also counts the number of sources which are up, using weights related to their production rate to discriminate between the two source types. Such function is denoted `buf+nSu`, which stands for “buffer occupancy plus number of sources up.” The specific expression we used to compute the importance is `buf + src1*lambda1 + src2*lambda2`. We also implemented a third strategy, counting the number of sources which are “down.” This last function is denoted `buf+nSd`, and the expression we used to compute the importance is `buf + NSrc1 + NSrc2 - src1 - src2`.

A brief reflection on the choice of these ad hoc importance functions is due before presenting the average convergence times they yielded. Including the state of the sources when computing the importance of a system state sounds sensible: the more sources producing packets, the faster a full occupancy should be observed. That suggests that the more operational sources it has, the more important a state should be deemed, which points at `buf+nSu` as a good I-FUN alternative. Why then try `buf+nSd`, which uses an opposite

[†] Rather than individual sources we use counters `src1` and `src2` of range 6 each, since N active sources of type i imply a buffer income rate equal to $N\lambda_i$. Thus the state space grows linearly with the number of sources.

heuristic? The answer is a stark “why not? we may just try.” This queueing system is the most complex introduced so far. The mixed open/closed queue from the previous section is simpler, yet it showed a behaviour not so easy to foresee, when changing the value of a single parameter. Perhaps some hidden subtleties in the interrelationship among components makes the heuristic behind `buf+nSu` to derive in a bad implementation of RESTART.

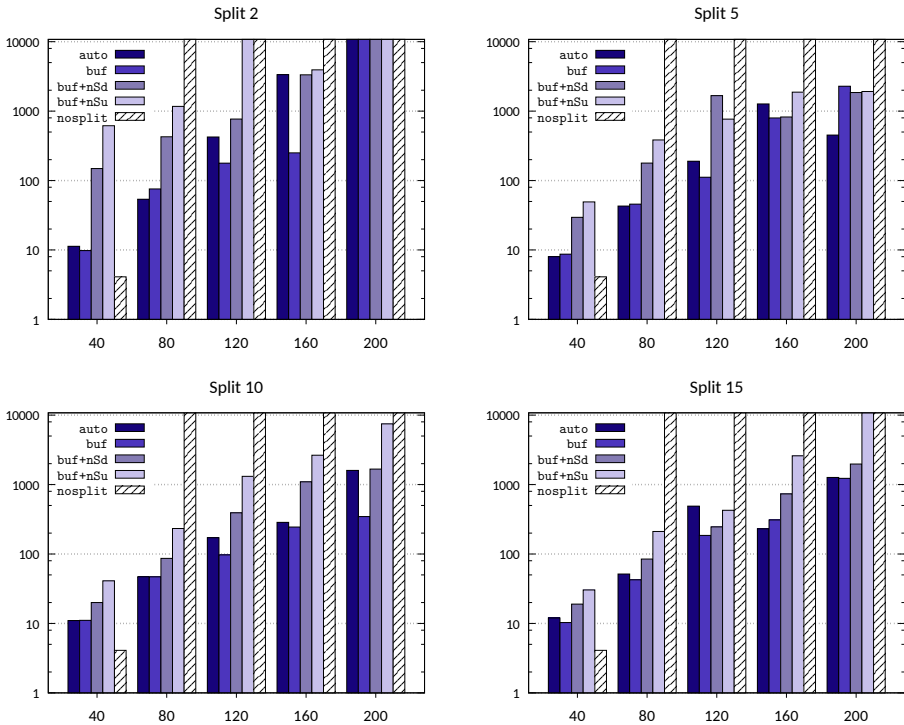


Figure 3.9: Transient analysis times of queueing system with breakdowns

Figure 3.9 presents the results from experimentation. Standard Monte Carlo simulations converged in time only for the smallest buffer size, $K = 40$. However, in doing so they outperformed all importance splitting simulations. This is not so surprising since $\gamma > 4.5e-4$ for that buffer size, comprising the *least rare event* studied so far. Moreover, in half of the cases only one threshold was selected by BLUEMOON, and in most of the other half of the cases, two thresholds were selected. Compare this to the 13–18 thresholds discussed in Section 3.5.4, and the 3–18 thresholds discussed in Section 3.5.2. Recall the gain derived from using RESTART is exploited when the splitting

is performed at the thresholds. When the event is not so rare, Algorithm 2 selects few importance values as thresholds, and the overhead of I-SPLIT (e.g. computing the importance of a state at each simulation step) belittles its gain. We believe that is the most likely reason why `nosplit` simulations outperformed the ones using RESTART for $K = 40$.

Studying the performance of the different importance functions, it can be seen that `auto` and `buf` converged the fastest in almost all settings. This defies the previous speculations around the real importance of the system state, and how it may be affected by the up/down state of the sources.

It is nonetheless unclear whether the state of the sources should be ignored when computing the importance. Maybe their influence ought to be scaled down by some *unknown factor*. Equivalently, the value of `buf` might need to be scaled up in the expressions of `buf+nSu` and `buf+nSd`, to emphasise a higher relevance of the buffer occupancy w.r.t. the states of the sources.

We base such hypothesis on the production and fail/repair rates of the sources, which are almost two orders of magnitude lower than the service rate $\mu = 100$. This means packets are quickly dequeued from the buffer, and thus observing many of them enqueued is against the odds. In particular, observing such high occupancy in the buffer may be far more important than having one more or less active source.

Furthermore, the number of operational sources is *scaled up* by the production rate λ_i in the expression of `buf+nSu`. For instance, 20 packets in `buf` and 3 sources up (of type 2) is deemed as important as 14 packets in `buf` and 4 sources up. This could be yielding a computed importance transverse to the real importance, with the increment in variance this implies.

To illustrate the last remark and its influence in the convergence times, consider a state where `buf=k-1` and one source of type 2 is broken. Suppose the importance function `buf+nSu` is used. Suppose also that the current system importance is i , and that the importance values $i + 1$ and $i + 6$ were selected as thresholds. Then for splitting value $k \in \mathbb{N}_{>1}$, a simulation which repairs the source and then enqueues a packet, produces k^2 more rare events than one that just enqueues a packet. These kind of scenarios augment the variability between the outcomes of the RESTART runs, increasing the computation times to convergence.

Last in this line of analysis, we draw attention to `buf+nSd`, which performed better than `buf+nSu` in all but one of the configurations tested. This in spite of the seemingly unreasonable heuristic behind `buf+nSd`, emphasising once again the difficulties of choosing a good I-FUN. Anyhow and in correspondence with the goals of the thesis, all conjecturing and reviewing can be dropped

when an algorithm to derive a reasonable function is available.

In particular, we highlight the good quality of the `auto` importance function via the ranking of functions for the different buffer capacities. From the average times to convergence presented in Figure 3.9, the importance functions which performed better for $K = 40, 80, \dots, 200$ were respectively `auto` with splitting 5 (8.0 s); `buf` with splitting 15 (42.6 s); `buf` with splitting 10 (97.1 s); `auto` with splitting 15 (230.5 s); and `buf` with splitting 10 (346.4 s). Moreover, in the three cases where `buf` was the winner, `auto` was the clear and close runner up. For instance, it took 42.9 s to converge with splitting 5 for $K = 80$, i.e. 30 ms (0.7 %) longer than `buf`.

3.6 Limitations of the monolithic approach

This chapter presented an automatable approach to perform model analysis by simulation, employing multilevel splitting to boost the convergence speed of the estimation mechanisms. The only inputs required are a model of the system, the property query specifying which transient or steady-state analysis to perform, a confidence criterion or an execution budget to meet (or both), and a global splitting value. If we resort to the splitting usually suggested for RESTART, which is the default in BLUEMOON[†], this input is the same that standard analysis by Monte Carlo simulation would require.

The previous section gives empirical validation of the relatively good efficiency that can be obtained using this approach. In all the systems and for a vast majority of their configurations, the automatic strategy from Section 3.3 yielded an I-SPLIT implementation that greatly outperformed the standard Monte Carlo approach. As desired, this performance difference was exacerbated by the rarity of the event: the smaller the probability to estimate, the better RESTART behaved w.r.t. standard simulations.

Moreover, the importance function automatically derived by BLUEMOON using Algorithm 1 rivalled the best ad hoc alternatives tested, most of them suggested as sensible or optimal choices by the authors of the articles from which the systems were extracted.

Unfortunately, this success is not general. BLUEMOON was designed on top of a model checker which, for the concerns of this thesis, studies Markov chains only. This is however no theoretical bound, since both algorithms

[†] In [VAVA02, VAVA06, VAVA11] the authors suggest choosing level-up probabilities (Definition 10) $p_i \approx 0.5$, which in our setting implies a global splitting of 2.

and also the automatic technique as a whole is oblivious of the memoryless property. Rather, the restriction limiting the applicability of the monolithic approach stems from a more practical concern: it is the same state space explosion problem that haunts the foundations of model checking.

This is not directly related to the use of importance splitting. RESTART simulations require, at most, the execution history that led to each state. The issue comes from the technique used to automatically build the importance function. Algorithm 1 needs an explicit representation of the full state space of the model, to store the importance values computed. Even worse, it also needs to traverse the transition matrix, whose likely sparsity helps little.

It is easy to appreciate the gravity of such requirements under the light of real life applications, which may need hundreds of modules to define a system. This renders infeasible any explicit representation of the state space, let alone the transition matrix. However it is not necessary to resort to the real world in order to meet the boundaries of the monolithic approach. The simplified model of a database facility in the following example is proof of it.

Example 7: Database system with redundancy.

Consider a database system consisting of disks arranged in clusters, disk controllers, and processors. For redundancy R the system is composed of two types of *processors* (with R copies of each type), two types of *disk controllers* (with R copies of each type), and six *disk* clusters (with $R + 2$ disks each). Figure 3.10 shows a schematic representation for two redundancy values: Figure 3.10 (a) depicts a system where $R = 2$ and Figure 3.10 (b) one where $R = 4$.

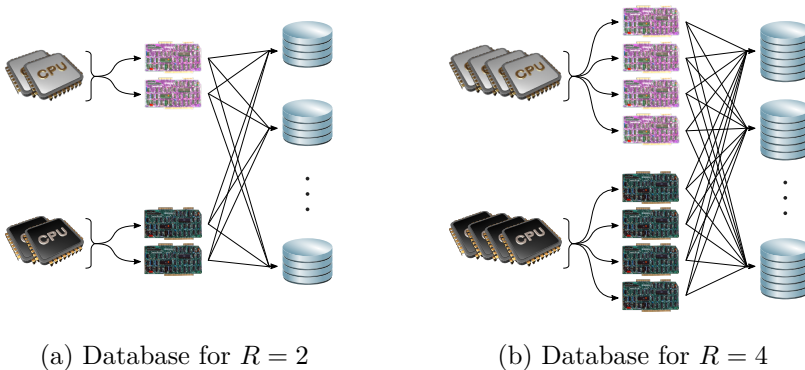


Figure 3.10: Database system with redundancy

Units lifetime is exponentially distributed with failure rates μ_D , μ_C , and μ_P for disks, controllers, and processors respectively. When a processor of one type fails, it causes a processor of the other type to fail with certain probability. Also, a unit can fail in mode 1 or mode 2 with equal probability, and each mode has its own repair rate. The system is considered operational as long as less than R processors of each type, R controllers of each type, and R disks on each cluster, have failed.

This system was originally studied in [GSH⁺92] using importance sampling, and then in [VA98, VA07a, VAVA11] with RESTART using importance functions defined ad hoc. The interest lies in studying the steady-state unavailability the system, i.e. where γ reflects the proportion of time the database is not operational. We focus on the setting used in the RESTART articles, where $(\mu_D, \mu_C, \mu_P) = (1/6000, 1/2000, 1/2000)$, the inter-processor failure probability is $1/100$, and the failures modes 1 and 2 have repair rates of 1 and 0.5 components per time unit respectively.

In [VA98] the author performs studies for redundancies greater than two, namely $R \in \{2, 3, 4\}$, and observes how I-SPLIT performs better for the higher values of R . As discussed in [BDH15] and later in [BDM17], such observations are reasonable since the underlying adjacency graph is highly connected, and R steps can make a system with no failed units become non-operational. Thus small values of R provide a meager setting for the splitting strategy, which relies on an efficient layering of the state space, stacking up between the initial state and the rare set. In short: the higher the redundancy, the better RESTART (and multilevel splitting in general) should perform.

Consider now the model of such system presented in Appendix A.5. Even though the description is modular, the variables forming the state space are global. Also, the same Markovian trick used in the queueing system from Appendix A.4 is employed, grouping the state of potentially individual modules instead of implementing them separately.

In spite of all this techniques, the resulting model has four variables of range $R + 1$ and six of range $R + 3$, plus the Boolean variable `f` in module `Repairman` to distinguish between the two failure types. So the total number of states is $2(R + 1)^4(R + 3)^6$. For redundancy two, the lowest one considered, this adds up to 2531250 states in a system with a dense transition matrix, for which PRISM reports 57825000 transitions.

This will quickly run into trouble regarding physical memory availability,

where the exact breaking point depends on the total memory available and the internal data types used. With PRISM development version 4.3 and 8 GiB of available RAM, the model checker throws an `std::bad_alloc` exception for $R = 4$, which would have had $2 \cdot 5^4 \cdot 7^6 = 147061250$ states and a reported 3774852200 number of transitions. \square

Example 7 illustrates a practical limitation of the monolithic approach presented in this chapter. The example also hints at another issue, more subtle and not critical in the sense that it does not impede us to apply the general strategy, yet with a clear negative impact on its efficiency.

This subtler issue is rooted in the high connectivity of the adjacency graph inherent to the database system. The efficiency of multilevel splitting increases as more levels are placed between the initial system state and the set of rare states. When few transitions can take a simulation path from any state to any other state, a rich layering into importance levels is infeasible.

For redundancy R , the database can move from the initial (fully operational) state to a rare state by taking R transitions. Therefore, the auto importance function computed by BLUEMOON yields only $R \in \{2, 3\}$ importance levels in the configurations tested, meaning a maximum of 1–2 thresholds. That seems hardly enough to get the full gain from I-SPLIT: recall the discussions for the queue with breakdowns from Section 3.5.5, where standard Monte Carlo simulations converged faster than RESTART simulations for the less rare setting. Something similar happens when running BLUEMOON on the model of Appendix A.5 for redundancy values $R \in \{2, 3\}$.

The rarity of the event studied in Example 7 is based on the very low probability of choosing a few transitions. These scenarios are detrimental for importance splitting, as mentioned at the end of Section 2.4, and usually better handled by importance sampling (when applicable).

On those lines it might be argued that the database system is inherently *flat*, and that R importance levels is the best we can do. Nevertheless that is not entirely true: compare a system with just a single disk failed, against a system where there is one failed component of each type (one disk per cluster, one CPU of each type, one controller of each type). It is much more likely to observe a rare event in the second case, since any further failure of any other component will cause a system failure. Still, the auto importance function computed by BLUEMOON deems both cases equally important, since both are (in truth!) one transition away from the set of rare states.

Algorithm 1 cannot distinguish between these cases because it operates on the fully composed model, where the *modular structure* is amalgamated into

a unified, highly connected system. A solution would require us to exploit such modular structure, understanding the contribution of each component to the global rare event, and using that to build the I-FUN.

Last and to conclude this chapter, let us return to the main concern regarding the monolithic approach. The inability to avoid *the state explosion problem* comes from the assumptions of Algorithm 1, which builds the automatic I-FUN on top of a single module. If a modular formalism like the PRISM input language is used, Algorithm 1 requires the composition of all the modules of the system, which generates a model whose state space grows exponentially with the number of modules involved.

A distributed approach sounds like the best solution, where we would apply (some variant of) Algorithm 1 locally on each module. The difficulty lies in achieving this without losing track of the global behaviour, since the north of the I-FUN is the *global* rare event. Therefore, when computing a function *local* to each system component, we must consider how does such global rare event reflect on each module. Otherwise we would be unable to choose the importance for its set of local states.

We have thus identified two challenges: distributing the monolithic approach from this chapter, honouring the global rare event while building the importance functions locally on each system component; and minding the modular structure of the system, to favour the layering of the state space and increase the gain of using importance splitting.

Those two challenges are the main subject of the following chapter, where we propose strategies to attack the problems, and methods to automate their implementation.

Automatic I-SPLIT: compositional approach

4

This chapter introduces strategies to adapt the automatic techniques of Chapter 3 to fit a compositional (or modular) description of the system. This is another main contribution of the thesis, involving a decomposition of the global rare event to build importance functions locally in each module, and then merging this distributed information to compute the *global importance* required by the splitting techniques.

Discussions start with the decomposition of the property describing the rare event. The aim is to project a *local rare event* inside of every module, from which a *local importance function* can be derived. Then, we study different ways of computing the importance of the global system model, using the local importance functions as building blocks. A recently introduced formalism for modelling general stochastic processes (time-homogeneous and free of nondeterminism) is considered. Also, another software tool is presented, which implements the strategies and algorithms introduced, using the aforementioned new formalism at its core. Finally the efficiency of the approach is demonstrated by means of case studies.

4.1 The road to modularity

The need for an explicit representation for the state space of the fully composed model is the most prominent limitation of the monolithic approach from Chapter 3. This requirement compromises the feasibility of the strategy, as exposed by the database system introduced in Example 7.

To avoid such problem, consider the naïve solution of applying the I-FUN derivation technique separately on each component of the system. This would only require the explicit state space representation of each *module*, but not of their composition. In particular for the database system, each component can either be failed or operational. Consider thus a model of it for redundancy $R \geq 2$, where each component is implemented as an independent module. Then applying this simplistic solution would need

$6(R + 2) + 2R + 2R = 10R + 12$ independent representations of binary state spaces. Hardly a memory issue even for e.g. $R = 100$.

Nevertheless the complications hiding behind such strategy are not so straightforwardly solved. The “I-FUN derivation technique” to which the naïve solution refers is nothing other than Algorithm 1, which takes two inputs: the module \mathcal{M} and its set of rare states A . To apply such algorithm on an individual module \mathcal{M}_i which forms part of a compound system, we must first identify its set of *local rare states*, A_i .

Identifying the sets $\{A_i\}$ in all modules is by no means trivial, at least not in the general case. Consider for instance a tandem queue where the user requests a steady-state analysis of the rare event

$$q_1 > \frac{C}{2} \Rightarrow q_2 = C$$

where \Rightarrow stands for the usual logical implication. Studying the module of the second queue, where the local variable q_2 is defined, it is unclear whether the concrete states associated to $q_2 < C$ can be regarded as rare. They do satisfy the rare event property, but only if $q_1 \leq C/2$. Which is thus the set of local rare states in the module of the second queue?

This brings forward the first challenge in the road to modularity:

- (a) projecting the global rare event onto the state space of each module, to guide the construction of local importance functions.

Suppose, for the sake of argument, that challenge (a) has been satisfactorily solved. Resuming the analysis of a solution, recall that all multilevel splitting techniques work with the importance of the states of the model, i.e. with the importance of the *global states*. In that sense the naïve strategy introduced above is incomplete, because it yields a set of functions which are interpreted separately on the *local states* of the modules. For each configuration of the global system this results in a set of importance values. Such set of importance values *needs to be merged somehow* in order to compute the global importance needed by I-SPLIT.

Persisting with naïvety, suppose we define such global importance as the summation of all local importance values. Consider a system where the contribution of the modules to the rare event is heterogeneous. We have already studied one such case: the queue with breakdowns introduced in Example 1 and experimented with in Section 3.5.5. Analyses suggested that the number of packets enqueued in the buffer—say, the importance of the module representing the buffer—was far more relevant in terms of

true (global) importance than the up/down state of the sources—say, the importance of the modules representing the sources.

Thus, the naïve solution of blindly adding up the importance of all modules is unsatisfactory in the general case. When merging the set of local importance values into a global importance, the degree of contribution of each module to the global rare event becomes relevant. All of which raises the second challenge in the road to modularity:

- (b) building a global importance function on top of the information stored locally on each module, considering the role of each module on the rare event.

Challenges (a) and (b) represent the main theoretic difficulties between the approach from Chapter 3 and a distributed version of it. Challenge (a) is analysed in further depth in Section 4.2, where a simple and robust algorithmic solution is proposed. Challenge (b) is the subject of Section 4.3.

4.2 Local importance function

As we have seen, finding a way to modularly decompose the approach from Chapter 3 is no straight road. The first challenge one finds in this direction is the identification of the rare event in the local state space of each module. In this section we propose an automatable strategy to tackle with such issue.

4.2.1 Projecting the rare event onto the modules

Modern modelling formalisms, like the input language of the PRISM model checker, are usually based upon a compositional description syntax. This means they allow the user to express the model of the system as the parallel composition of a set of smaller *system modules*. Each module has its own behaviour and it can usually define its own set of *local variables*, whose scope does not transcend the module where they are defined.

In spite of this locality of scope, the expression of the user query for the (global) rare event can include variable names from any module. Consequently, the property conveyed by such expression has semantics on the global system model, and not on the modules taken individually. In other words, the rare event property is interpreted as a set of *global* states, possibly describing specific simultaneous configurations of many modules.

The first step towards a modular approach is to identify, in every module taken individually, the set of local states A_i corresponding to the global rare event. The non-trivial question is how to interpret the *global property*—given by the user as a rare event query—locally on each module.

To illustrate the fundamental difficulty consider the tandem queue network from Example 2, described by a compositional model like in Code 3.3. Compare the following definitions of the event under study, to use in a steady-state analysis of the model:

- (a) $q_2 = C$,
- (b) $q_1 = C \wedge q_2 = C$,
- (c) $q_1 \geq C/2 \Rightarrow q_2 = C$.

All three definitions speak of a saturation in the second queue (**Queue2**), but the role of the first queue (**Queue1**) is harder to grasp.

Variable q_1 is missing from definition (a), so the module of the first queue could be ignored when deriving the local importance functions, since it does not change the validity of the formula $q_2 = C$. In other words, for definition (a) the local importance function of **Queue1** could be $null \doteq \lambda x . 0$.

Definition (b) does include variable q_1 . More precisely, given the logical expression is a *conjunction* of two terms, the occurrence of q_1 in one of those terms implies it is a key component of the global rare event. This indicates that the local importance function of the module of the first queue will not be *null*, contrary to what happened with definition (a). Moreover, the local rare states in **Queue1** must be those satisfying $q_1 = C$.

Finally, definition (c) deserves a deeper analysis. That formula is equivalent to $\neg(q_1 \geq C/2) \vee q_2 = C$. Therefore, the local rare states in **Queue1** are those which *do not satisfy* $q_1 \geq C/2$. This is at odds with situation (b), where the term containing q_1 was used as it occurs in the definition. Here instead the local rare states are identified by the *negation* of the term where q_1 occurs.

In general, the difficulty lies in knowing whether to take positively or negatively the occurrence of a variable in the rare event. The whole expression can have several levels of nested negations, entangling matters. Hence when analysing a module, it is unclear how to interpret the subformulas where its variables appear. For instance, in the previous example with the module of the first queue, the occurrence of q_1 in a comparison must be taken positively for definition (b) of the rare event, and negatively for definition (c).

Reviewing the analysis applied in case (c), notice that the conclusion of using $\neg(q_1 \geq C/2)$ was reached by means of the equivalence

$$q_1 \geq \frac{C}{2} \Rightarrow q_2 = C \equiv \neg \left(q_1 \geq \frac{C}{2} \right) \vee q_2 = C .$$

Specifically, the *simpler* side of the equivalence, viz. the one with disjunction and negation as logical operators, states more clearly how to interpret the logical expression for marking the local states.

Similarly, consider the rare event expression $\neg(q_1 \neq C \vee q_2 < C)$. It seems best to use the equivalent expression $q_1 = C \wedge q_2 \geq C$ when interpreting which states should be marked as rare.

The broad idea is to transform the rare event formula into an equivalent expression in some normal form, where all nesting has been solved and there are no logical operators of implication nor equivalence. In that respect the *disjunctive normal form* (DNF) is a good candidate. A DNF formula is a disjunction of *clauses*, each of which is a conjunction of *literals*. A literal is an atomic proposition or the negation of one, i.e. a Boolean variable or the comparison of numeric expressions involving numbers and variables.

Using formulae in DNF has several advantages:

- it is a standard for formula representation, with the usual implications this conveys, e.g. knowledge from the side of the reader can be assumed;
- all propositional formulae can be equivalently expressed in DNF, so there is no restriction on what can be analysed;
- the rare event can be clearly identified as the satisfaction of *any* of the clauses composing the expression;
- there are no nested negations and thus no ambiguity when interpreting a literal, since each one clearly states how (positively or negatively) it contributes to the rare event.

From the three examples above the first two are already in DNF: definition (a) is a single literal, and (b) is a single clause composed of two literals. Definition (c) is not in DNF: $\neg(q_1 \geq C/2) \vee q_2 = C$ is an equivalent DNF formula composed of two clauses, each with a single literal.

The main advantage of dealing with formulae expressed in DNF is that the literals carry all the information regarding how to interpret the occurrence of a variable. Therefore a simple *projection* of the formula onto the namespace of each module provides a correct and unambiguous identification of the local

rare states. In that sense the literals of a formula in DNF can be regarded as its building blocks, and hence considered indivisible during a projection. This means e.g. projecting $q_2 > C$ onto the scope of `Queue1` yields an empty expression—as explained in the next section—which is not used to identify the local rare event; because even though the constant C (e.g. the identifier `c`) has global scope and is thus known by `Queue1`, the variable q_2 (e.g. the identifier `q2`) exists *only* within the namespace of `Queue2`.

Continuing with the previous example of the tandem queue, consider a projection of definition (a), which is already in DNF, onto the namespace of the module of the first queue. Since `q2` is outside the scope of `Queue1`, the projection will yield an empty expression. This suggests there is little or no information regarding the rare event in such module, which could thus be safely omitted from importance computation. Contrarily, the same projection onto the namespace of `Queue2` yields the expression $q_2 = C$ (e.g. `q2=c`). This will correctly identify the concrete states corresponding to a saturated second queue as the local rare states.

In the case of definition (b), which is also in DNF, a projection into the namespace of the module of the first queue will result in $q_1 = C$. This means the local rare states are those corresponding to full occupancy in the first queue, as desired. So in this case `Queue1` is relevant for importance computation. The situation for `Queue2` is the same as in (a).

Last, consider the DNF formula $q_1 < C/2 \vee q_2 = C$, equivalent to definition (c) of the rare event. A projection into the namespace of `Queue1` yields $q_1 < C/2$, identifying as local rare states those corresponding to a first queue occupied to less than half its capacity. As previously discussed, this is the desired result, and could be reached without further processing because the expression used was in DNF. The situation for `Queue2` is the same as for the other two definitions of the rare event.

4.2.2 Algorithms and technical issues

The previous section stated that the logical expression of the user query will be in disjunctive normal form, yet so far the projection of the formula on the local scope of each module has been informally introduced. We need an algorithm to automate this procedure.

Projecting a literal which contains an identifier out of scope must yield an *empty expression* with no effect in importance computation. This means that in the logical expression (of local scope) resulting from our algorithm, such literal must appear as the neutral element of the logical operator for

which it is an operand. Since the neutral elements of the disjunction and the conjunction are different, the projection algorithm must tell these cases apart. On the one hand, a projected literal which becomes empty inside of a clause should be replaced with \top . On the other hand, a full clause becoming empty when projected must be replaced with \perp .

Furthermore, if the full rare event expression becomes empty when projected on a module, e.g. because all occurring literals contained variable names out of scope, the resulting expression will be \top . This means all the local states of an uninteresting module will be considered rare. The reason behind this is related to the later composition of the local importance functions; it will be justified in Section 4.3.

Algorithm 3 presents a procedure to perform the projection just described. Notice that there is a finite number of clauses for each DNF formula φ , each containing a finite number of literals. This means both loops in the algorithm will iterate a finite number of times. Routine `free_vars(σ)` finds the names of the free variables occurring in the literal σ . Given each literal has finite length the routine will terminate in finite time. Furthermore, routines `global_vars()` and `\mathcal{M} .vars()` return the names of the variables within the global scope and the scope of module \mathcal{M} respectively. Since there is a finite number of variables to any system model those routines terminate, in time linear on the number of system variables. In view of the above, no formal proof of termination is required for Algorithm 3.

This algorithm chops the global expression of the rare event, producing smaller formulae which can be independently interpreted in the scope of each system module. Notice however that this projection policy *rules out diagonal cases*, i.e. situations where the rare event involves operations with variables from different modules in direct comparison.

The issue is unavoidable as far as the arithmetic comparison operators are concerned, but can be circumvented in other situations. For example, suppose the user requests a transient analysis of the tandem queue for the rare event

$$\min(q_1, q_2) \geq \frac{C}{2}.$$

Interpreting the relationship between operators $\min(\dots)$ and \geq we see that the rare event expression is equivalent to $q_1 \geq C/2 \wedge q_2 \geq C/2$, to be projected as $q_1 \geq C/2$ and $q_2 \geq C/2$ onto modules `Queue1` and `Queue2` respectively. It seems clear that this is the desired behaviour.

Algorithm 3 Projection of a DNF expression onto a module

Input: module \mathcal{M}
Input: (global) rare event expression in DNF φ

```

 $\tilde{\varphi} \leftarrow \perp$ 
for all clause  $\psi \in \varphi$  do
   $\tilde{\psi} \leftarrow \top$ 
  for all literal  $\sigma \in \psi$  do
    if  $\text{free\_vars}(\sigma) \subseteq \mathcal{M}.\text{vars}() \cup \text{global\_vars}()$  then
       $\tilde{\psi} \leftarrow \tilde{\psi} \wedge \sigma$ 
    end if
  end for
  if  $\tilde{\psi} \neq \top$  then {syntactic comparison}
     $\tilde{\varphi} \leftarrow \tilde{\varphi} \vee (\tilde{\psi})$ 
  end if
end for
if  $\tilde{\varphi} = \perp$  then {syntactic comparison}
   $\tilde{\varphi} \leftarrow \top$ 
end if

```

Output: local rare event expression $\tilde{\varphi}$

Suppose instead that the global rare event is

$$q_1 > 0 \wedge \frac{q_2}{q_1} \geq 2 \Rightarrow q_2 \geq C,$$

which queries the probability of a saturation in the second queue, when it has at least twice as much packets as the first queue. An equivalent DNF formula is $q_1 \leq 0 \vee q_2 < 2q_1 \vee q_2 \geq C$, where q_1 and q_2 appear *in direct comparison* in the second clause. The projection of the literal $q_2 < 2q_1$ will yield an empty expression for both modules of the tandem queue, because q_1 is out of scope for `Queue2` and q_2 is out of scope for `Queue1`. Thus, the identification of the local rare states will be given by $q_1 \leq 0$ in `Queue1`, and by $q_2 \geq C$ in `Queue2`, which is at odds with the original user query. In particular, such projection does not reflect the dependence between the saturation in the second queue and the occupancy at the first queue.

Be that as it may, we do not consider this as a major problem due to conspicuous pragmatic reasons. From a purely practical perspective,

throughout our research we have encountered few studies involving these kind of properties. Most articles deal with simple definitions of the rare event involving a single system variable. Also, on a slightly more theoretical appreciation, when a direct comparison of variables is necessary then the semantics behind them is typically related. This means that such variables will likely be defined in the same semantic unit, viz. the same module.

In spite of those reasons there are examples of diagonal cases in the rare event literature. For instance [VAVA06] study the transient behaviour of the tandem queue for the rare event $q_1 + q_2 \geq C$. The only workaround in such cases is to define the variables affected within the same module, because Algorithm 3 will otherwise yield empty projections as discussed. For the tandem queue this results in a monolithic representation like the one from Appendix A.1. We highlight however that in complex systems with several components it is unnecessary to merge them all into a single monolithic model. Composing the modules with the variables in direct comparison will suffice.

Once all projections are ready we can proceed to build the local importance function of each module. When the projection is empty for a component we assume it does not play a primary role in the rare event, and render the local importance function *null*. Otherwise we use the formula resulting from the projection, which is in DNF by construction. The projected formula $\tilde{\varphi}_i$ is used to identify the local rare states A_i in module \mathcal{M}_i . Then \mathcal{M}_i and A_i are provided as input to Algorithm 1, which yields the local importance function of the i -th module.

Algorithm 4 presents the full procedure. Routines `project(...)` and `derive_importance_function(...)` are respectively Algorithms 3 and 1. The member function `identify_states(...)` of each module \mathcal{M}_i returns the set of local concrete states identified by its argument. Since the identification takes place within the local scope of the module, the formula $\tilde{\varphi}_i$ we feed it with must contain only variables which are global or local to \mathcal{M}_i . Routine `project(...)` ensures this is the case.

Algorithm 4 clearly terminates since the number of modules is finite, and routines `project(...)` and `derive_importance_function(...)` are algorithms for which a proof of termination has already been given.

It is important to highlight that whenever the projection of the DNF formula is empty, the nature of the *null* function assigned to f_i will in truth depend on the *composition operator*. In a nutshell, *null* must return the neutral element of an arithmetic operator, which e.g. for $+$ means f_i will be

Algorithm 4 Local importance functions computation

Input: modules set $\{\mathcal{M}_i\}_{i=1}^m$

Input: global rare event formula φ

```

assert_DNF( $\varphi$ )
for all module  $\mathcal{M}_i$  do
   $\tilde{\varphi}_i \leftarrow \text{project}(\mathcal{M}_i, \varphi)$ 
  if  $\tilde{\varphi}_i = \top$  then                                     {syntactic comparison}
     $f_i \leftarrow \text{null}$ 
  else
     $A_i \leftarrow \mathcal{M}_i.\text{identify\_states}(\tilde{\varphi}_i)$ 
     $f_i \leftarrow \text{derive\_importance\_function}(\mathcal{M}_i, A_i)$ 
  end if
end for

```

Output: local importance functions set $\{f_i\}_{i=1}^m$

$\lambda x.0$, whereas for $*$ it will be $\lambda x.1$. This matter is explained in further detail in the following section.

Therefore, using Algorithms 3 and 4 we can compute and store the importance of the system states in a per-module basis, whose physical memory requirements grow *linearly* with the number of modules composing the model—though exponentially on the number of global system variables. This in contrast to the monolithic I-FUN construction of Chapter 3, whose representation as an array in the memory of the computer grows *exponentially* with the number of modules. The result is a notion of local importance functions, the set $\{f_i\}_{i=1}^m$ produced by Algorithm 4, which still need to be combined with each other in order to compute a global importance function.

4.3 Composition of the local importance functions

In Section 2.6, the description of RESTART is oblivious of the way in which the importance function is computed or stored. During a simulation, RESTART simply needs the importance of the current (global) state after taking a transition. The same transparency is required by all known multilevel splitting techniques and even by Algorithm 2 to select the thresholds. Therefore, the approach from the previous section must be complemented with some

procedure to decide the importance of each state of the fully composed model, taking as input the local importance of the modules.

4.3.1 Basic composition strategies

The simplest option to compose the local importance functions is letting the user settle the matter, who would specify an ad hoc way to combine (the local importance functions of) the modules. We say the user provides an ad hoc *composition function*. Formally, the input required is an algebraic expression using (numeric literals and) identifiers which correspond to the functions $\{f_i\}_{i=1}^m$.

Say e.g. we are experimenting with the modular description of the tandem queue from Code 3.3, and let `Q1` and `Q2` stand for the the local importance functions of modules `Queue1` and `Queue2` respectively. Then the user could specify composition functions such as `Q1+Q2`, `2*Q1+5*Q2`, and `(1+Q1)*(1+Q2)`. The choice will likely depend on the nature of the rare event under study.

This is comparable to an ad hoc specification of the importance function. Rather than asking the user to operate with variables, the arithmetic expression provided is of higher level, dealing with whole modules. The local importance function built for a module is trustworthy, in the sense that it effectively translates the behaviour of the module into importance information—see Chapter 3. Hence using these local functions as building blocks of the expression, instead of the local variables of the modules, is an improvement over explicitly defining an ad hoc importance function.

There is no fundamental problem with such strategy, but the general goal of the thesis is to develop automatic ways to lighten the burden of the user, so an algorithmic solution is preferable.

The simplest automatic way of combining the local importance functions is choosing an associative binary arithmetic operator, a *composition operand*, and apply that to all functions. Natural candidates are summation, product, max, and min. Notice each operand has its own neutral element. Therefore choosing `+` will make Algorithm 4 use $\lambda x. 0$ as the *null* function; choosing `max` will use $\lambda x. -\infty$, and so on.

The performance of I-SPLIT will variate with the choice of operator, influenced by the nature of the model and of the rare event. That is evident since the local importance functions (i.e. the arguments of this composition operand) depend on the expression of the rare event.

For instance if the tandem queue is analysed for the rare event $q_2 = C$, using summation or `max` as composition operand yields the same global impor-

tance, since Q_1 will be a *null* local function and thus $Q_1+Q_2 = \max(Q_1, Q_2) = Q_2$. By contrast for the rare event $q_1 = C \wedge q_2 = C$, choosing max or summation yields different results, as shown in the following example.

Example 8: I-FUN compositions for the tandem queue.

Figure 4.1 shows importance functions on the concrete state space of a (CTMC) tandem queue with capacity $C = 3$, for the rare event expression $q_1 = C \wedge q_2 = C$. Moving from left to right in a lattice increases the number of packets in the first queue; moving from the bottom upwards increases the packets in the second queue. Arrows indicate the transitions of the system: an external packet arrival into the first queue is an horizontal arrow; a packet moving from the first to the second queue is a diagonal arrow; and a packet leaving the second queue is a vertical arrow.

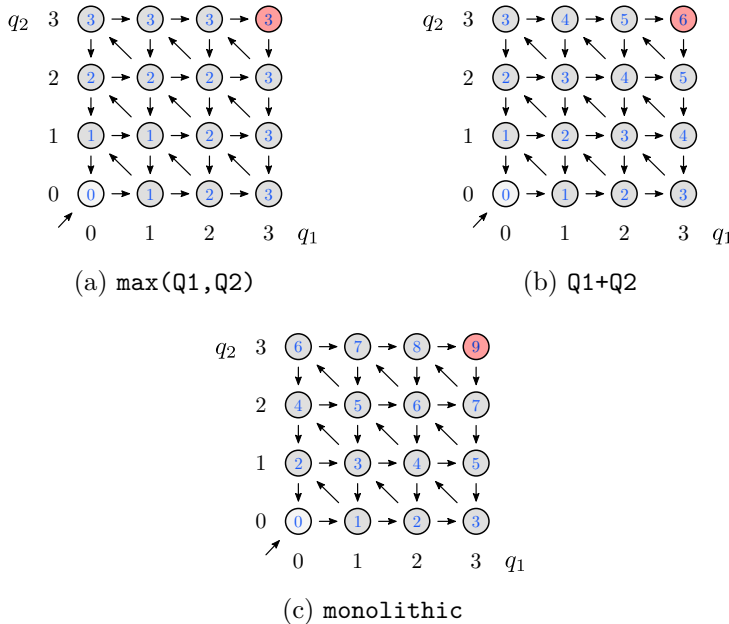


Figure 4.1: Importance functions for the tandem queue with rare event $q_1 = C \wedge q_2 = C$

Different importance functions are presented in the schemes. The num-

bers within the concrete states (i.e. within the nodes of the lattices) are the importance values each I-FUN assigns to the states. Figures 4.1 (a) and 4.1 (b) show modular functions built with the approach from Section 4.2. In Figure 4.1 (a) the composition operand \max is used, whereas in Figure 4.1 (b) summation is used. Figure 4.1 (c) shows the monolithic I-FUN that the approach from the previous chapter would construct.

The symmetry of the importance values in Figures 4.1 (a) and 4.1 (b) is a result of the compositional nature behind the assignment of the global importance values. Whereas the monolithic I-FUN from Figure 4.1 (c) can consider each concrete global state individually, functions built with the compositional approach can only distinguish between the states of a separate module. As a consequence there can be global behaviours which the monolithic approach can grasp but which are invisible to the eyes of the compositional approach. This is related to the diagonal cases mentioned in Section 4.2.2 and is further discussed in Section 4.3.2. \square

In previous sections we introduced the *monotonicity condition* of an I-FUN, to mean that every simulation following a shortest path from the current state to the rare set, will traverse a monotonically increasing sequence of importance values. For the setting depicted in Figure 4.1, a shortest path is one that only uses horizontal and diagonal arrows to reach the single rare state on the top-right corner of the lattice. Notice that the monolithic I-FUN from Figure 4.1 (c) satisfies the condition, as expected.

The monotonicity condition is a desirable property for an importance function. It ensures that simulation paths moving in the right direction will not suffer from truncation, which is clearly advantageous. However the same holds using a slightly weaker definition, requesting the visit of *non-decreasing* instead of *increasing* importance values. Under this new definition, the composition strategy $Q1+Q2$ from Figure 4.1 (b) in Example 8 also satisfies the monotonicity condition. That is not true for $\max(Q1, Q2)$ in Figure 4.1 (a), notice e.g. the diagonal transition $(q_1, q_2) = (2, 0) \rightarrow (1, 1)$.

All this suggests that summation is a better composition operand than \max when the rare event is $q_1 = C \wedge q_2 = C$. That is also indicated by the (much less rigorous) rule of thumb of comparing the maximum importance value assigned in each case. The sum of $Q1$ with $Q2$ gives importance 6 to the rare state of the system, whereas $\max(Q1, Q2)$ reaches only the value 3. Recall that for any model and rare event, the higher the importance range of a function, the more options Algorithm 2 has to choose thresholds from. From

this viewpoint $Q1+Q2$ is also more promising than $\max(Q1, Q2)$, since broadly speaking more thresholds mean more splitting and thus better simulation efficiency, at least when adaptive algorithms like Adaptive Multilevel Splitting or Sequential Monte Carlo are used to select thresholds intelligently.

4.3.2 Monolithic vs. compositional importance functions

In the approach from Chapter 3, the I-FUN is built on top of the concrete state space of the fully composed model. Algorithm 1 is used to perform the task and, as a result, the *global static behaviour* is exploited in the process. By static behaviour we refer to the transitions of the adjacency graph at concrete level, which are aware of the predecessors of a state but ignore the probabilistic/stochastic nature of the edges[†].

In this chapter the *local static behaviour* of each individual module is exploited separately. There can however exist interactions between the system components, which appear explicitly at global level but cannot be captured by Algorithm 1 at local level. If inadequate strategies are chosen to compose the local importance functions, this can deteriorate the quality of the resulting function when any of the following two situations arise:

- (a) the variables used in the rare event expression reside in a few small modules, which express little of the full process behaviour;
- (b) module synchronization subsumes much of the semantics of the full process, so the collective state of all system components affects greatly the behaviour of each individual component.

The following example illustrates the hazards of an inappropriate use of the compositional approach when the system exhibits those problems.

Example 9: Building houses of cards.

Lazing on a Sunday afternoon you poke about in grandpa's closet, God rest his good soul. Inside of a dusty suitcase you find an old pack of Spanish playing cards, and start building card houses. You build one house per suit, using all cards in the suit. You build them close together so that any misplaced card will tumble all houses down—whoops!

Code 4.1 shows a PRISM model of the game in a discrete time setting. Since it is a DTMC all edges have probabilities. These are implicitly

[†] As opposed to the *dynamic behaviour* of the system affecting e.g. a RESTART simulation.

equal to 1.0 but for the edge of lines 9 and 10, where equal probability is given to properly placing a card vs. tumbling the whole thing down.

Module `House` represents how many cards have been correctly placed while building the house with the current suit. Module `Suit` keeps track of the current suit under use; Spanish suits comprise *Copa* (cup, 🍷), *Oro* (gold, 🟡), *Basto* (club, ♣️), and *Espada* (sword, ⚔️).

Code 4.1: Card houses game


```

1 dtmc
2
3 const int NUM_SUITS = 4; // Copa, Oro, Basto, Espada
4 const int CARDS_IN_SUIT = 10;
5
6 module House
7     state: [0..2]; // 0:idle ; 1:place_card ; 2:tumble_cards_down
8     cards: [0..CARDS_IN_SUIT];
9     [] state=0 & cards<CARDS_IN_SUIT -> 0.5: (state'=1)
10        + 0.5: (state'=2);
11     [] state=1 & cards<CARDS_IN_SUIT -> (cards'=cards+1) & (state'=0);
12     [whoops] state=2 & cards<CARDS_IN_SUIT -> (cards'=0) & (state'=0);
13     [house] cards=CARDS_IN_SUIT -> (cards'=0);
14 endmodule
15
16 module Suit
17     suit: [1..NUM_SUITS];
18     [whoops] true -> (suit'=1);
19     [house] suit < NUM_SUITS -> (suit'=suit+1);
20 endmodule

```

Starting off easy the goal is to build a single house, say using the *Copa* suit. The question then rises, how likely are you to succeed in the first attempt, without any tumbling down of cards? This question regards transient behaviour, and the model from Code 4.1 allows a succinct property query to express it:

$$P=? [\text{state} < 2 \text{ U } \text{suit} = 2].$$

Let us compare the importance functions that the monolithic and compositional (with summation) approaches would generate. For that purpose Figure 4.2 shows a spartan representation of the concrete state space of the system. In the scheme only the *Copa* suit is represented, and variable `state` from module `House` is abstracted away. That variable exists to distinguish between a proper card placement and a tumble-down. In the schemes from Figure 4.2 that is represented with a forked arrow, , where the down-going tip stands for the tumble-down (whoops!), and the straight tip stands for the proper card placement.

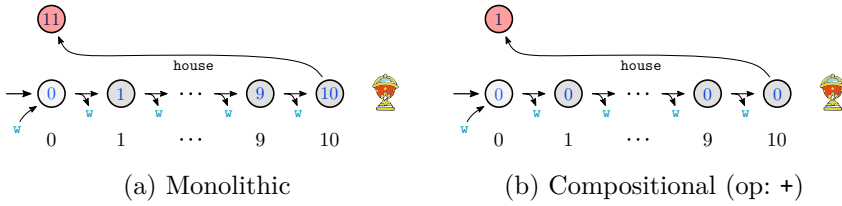


Figure 4.2: Card houses game (beginners version)

The flatness of the function in Figure 4.2 (b), which only has the importance values $\{0, 1\}$, is due to problem (a) mentioned before. Since the expression of the rare event in the property query is `suit=2`, the local I-FUN of module `House` is *null* because none of its variables appear in the expression. Oppositely, the monolithic approach can—only—observe the value of variable `suit` in interaction with variable `cards`, thus capturing the behaviour of the fully composed model.

The compositional approach yields a poor function because the expression of the rare event leaves out a relevant module, i.e. due to problem (a). In some cases an easy workaround is to force the occurrence of relevant yet inessential variables in the expression of the rare event. For instance the query `P=? [state<2 U suit=1 & cards=CARDS_IN_SUIT]` enriches the compositional approach in the previous situation, yielding the same importance function than the monolithic approach would.

Unfortunately problem (b) is harder to counter, since making the functions rigorously local to each module implies losing track of the nature of interactions with other modules. To illustrate the hazards of this problem consider the full version of the game, where you wish to build a house with each of the four suits. The property query is:

$$P=? [\text{state}<2 \ U \ \text{suit}=\text{NUM_SUITS} \ \& \ \text{cards}=\text{CARDS_IN_SUIT}] .$$

Problem (a) is thus avoided and applying the monolithic and compositional approaches yields the importance functions represented in Figure 4.3. As before, summation is used to compose the local importance functions of modules `House` and `Suit` in Figure 4.3 (b).

On the one hand Figure 4.3 (a) shows a natural continuation of the previous monolithic result presented in Figure 4.2 (a), extended here to consider the four suits of the Spanish pack.

On the other hand the I-FUN from Figure 4.3 (b) presents an anomaly

much detrimental to I-SPLIT: each time a house is completed using all the cards in a suit, the importance drops almost to the initial value. In multilevel splitting simulations, that might truncate several path offsprings which were actually moving in the right direction.

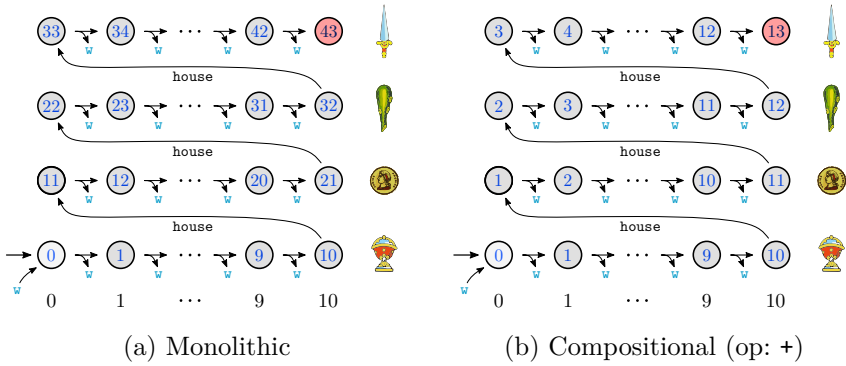


Figure 4.3: Card houses game (full version)

□

At least two factors contribute to the issue observed with the compositional approach depicted in Figure 4.3 (b). One of them is problem (b), i.e. the system from Code 4.1 uses the communication between modules as a key element to progressively evolve towards the set of rare states. On its own however that is not detrimental to the approach. The second factor that led to the poor result of Figure 4.3 (b) is using summation as composition operand. This clearly failed to apprehend the nature of the evolution towards the rare event.

Combined with a proper composition function, the compositional approach can mimic the monolithic I-FUN. Take for instance the function `CARDS_IN_SUIT*SUIT+HOUSE`, where `SUIT` and `HOUSE` stand for the local functions of the homonymous modules in Code 4.1. The reader can check this yields (almost) the same function than that of Figure 4.3 (a).

Yet falling back to the use of a composition function requests an ad hoc intervention by the user, which defeats the purpose of the thesis. An automatable procedure is desirable, with enough plasticity to behave well when the system modules exhibit non-trivial interaction schemes. Our proposal in that respect is the subject of the following section.

Example 9 may give the impression that whenever feasible, using the

monolithic approach from Chapter 3 will always yield an importance function superior to the ones that can be built with the compositional approach. That is most certainly not the case.

First, the model of the card houses game presented in Code 4.1 is somewhat artificial. It was devised with the intention of dislocating an otherwise purely sequential evolution from the initial state towards the rare event. The goal was to put in evidence that a naïve application of the compositional techniques from the previous sections can yield poor results.

Second, recall the limitations of the monolithic approach presented in Section 3.6. There were two issues with the database system from Example 7: a monolithic I-FUN would occupy more physical memory in the machine than available; and the model displays a flat structure when all its modules are composed. The compositional approach is primarily designed to tackle with the first issue, but is also very useful to attack the second one.

The specific problem was that once composed, the state space of the full database process distinguishes only R importance levels for redundancy $R \in \mathbb{N}_{>1}$. That leaves little room for an efficient layering of the state space, as needed by multilevel splitting. If instead the structure of the system is exploited prior to composition, like the compositional approach allows, more importance levels could be fabricated to boost the splitting needed by I-SPLIT. We elaborate on this in the next section.

4.3.3 Composing the functions with rings and semirings

The primary goal is thus to conceive an *automatic composition strategy* for the local functions, which performs well regardless of the specific inter-module interactions, resulting in a reasonable global importance function.

Remember the application of the compositional approach to the tandem queue in Section 4.3.1. Depending on the rare event under study, using the composition operands \max or $+$ could yield different global importance functions. In particular $\max(Q1, Q2) = Q1+Q2$ for the rare event $q_2 = C$, whereas Example 8 shows summation is the best candidate for the rare event $q_1 = C \wedge q_2 = C$. All of this suggests that the performance of a composition strategy is directly affected by the expression defining the rare event.

Consider now a triple tandem queue, i.e. packets processed by the server of the second queue are buffered in a third queue, leaving the system only after they are processed by the server of this third queue. Code 4.2 shows a model of the system.

Code 4.2: PRISM triple tandem queue model

```

1  ctmc
2
3  const int      c = 7; // Queues capacity
4  const int lambda = 1; // Arrival rate
5  const int mu1 = 2; // Server1 rate
6  const int mu2 = 4; // Server2 rate
7  const int mu3 = 6; // Server3 rate
8
9  module Arrivals
10     [arrival] true -> lambda: true;
11 endmodule
12
13 module Queue1
14     q1: [0..c];
15     [arrival] q1<c -> 1: (q1'=q1+1); // Receive
16     [arrival] q1=c -> 1: true;
17     [service1] q1>0 -> mu1: (q1'=q1-1); // Process
18 endmodule
19
20 module Queue2
21     q2: [0..c];
22     [service1] q2<c -> 1: (q2'=q2+1); // Receive
23     [service1] q2=c -> 1: true;
24     [service2] q2>0 -> mu2: (q2'=q2-1); // Process
25 endmodule
26
27 module Queue3
28     q3: [0..c];
29     [service2] q3<c -> 1: (q3'=q3+1); // Receive
30     [service2] q3=c -> 1: true;
31     [service3] q3>0 -> mu3: (q3'=q3-1); // Process
32 endmodule

```

Say the user requests a steady-state analysis in the model from Code 4.2 for the rare event

$$(q_1 = C \wedge q_2 = C) \vee (q_1 = C \wedge q_3 = C). \quad (15)$$

The formula is already in DNF; the projection on the modules of the queues will label as rare the local states satisfying $q_i = C$ for $i \in \{1, 2, 3\}$.

Given the states of all modules are relevant, it seems reasonable to believe that summation is a natural candidate for composition strategy, viz. employing the global importance function $Q_1+Q_2+Q_3$, where Q_i stands for the local importance function of module `Queuei`.

Notice however that eq. (15) is equivalent to $q_1 = C \wedge (q_2 = C \vee q_3 = C)$, hinting at a higher relevance of the number of packets in the first queue w.r.t. the number in the second or third queue taken on their own. That is

overlooked by the global importance function $Q_1+Q_2+Q_3$, which considers the number of packets in each queue equally relevant.

This suggests there is a very strong correlation between the *precise* rare event expression used, and the performance of the composition strategy. Thus, the possibility of deriving the latter using the former is quite appealing. The objective is to generate a composition strategy as tightly fit to the rare event expression as possible.

The requirement to work with logical formulae expressed in DNF already gives some structure that could be exploited. Recall the building blocks of DNF formulae are the literals, and the current focus is on literals consisting of variables local to the scope of a single module—see Section 4.2.2. We could hence map each literal in the rare event formula, to the local importance function of the module where the literal resides.

The above indicates that eq. (15) yields the sequence of local importance functions (Q_1, Q_2, Q_1, Q_3) . Those functions are to be composed *somehow* following the DNF structure of eq. (15), to build the arithmetic expression which will serve as global importance function. That sounds reasonable but leaves an open question: how should the disjunction and conjunction operators from the DNF expression be interpreted?

Notice that the previous reflection regarding the higher relevance of `Queue1` for eq. (15) is reached using the distributive property of \wedge with respect to \vee . In particular, the pair (\vee, \wedge) is an algebraic structure known as *ring*, where \vee and \wedge are respectively the *addition* and the *multiplication* of the ring. The distribution of multiplication over addition, e.g. of \wedge over \vee , is an axiom of the ring and semiring algebraic structures.

Therefore we propose to choose pairs of algebraic operators (\oplus, \odot) which present ring or semiring structure, and map the occurrence of (\vee, \wedge) in the DNF expression of the rare event to (\oplus, \odot) in the composition strategy. In particular, maximum and summation—i.e. $(\max, +)$ —and summation and product—i.e. $(+, *)$ —have respectively semiring and ring structure.

So for instance the rare event from eq. (15) can be turned into the composition strategy $\max(Q_1+Q_2, Q_1+Q_3)$, which by the distributive property of $+$ with \max results in the global importance function $Q_1+\max(Q_2, Q_3)$. Remember Q_i does not symbolise a variable of a module, but is rather interpreted as the local importance function of a module.

To contrast the performance of this strategy against the simplistic approach of employing a binary associative operand, consider a triple tandem queue where the current global state is $(q_1, q_2, q_3) = (C - 1, C - 1, 0)$.

Using summation, i.e. the composition operand $+$, results in the global

importance function $Q1+Q2+Q3$. This function misses out the structure that appears in the expression of the rare event, and for instance yields the same importance as a new incoming packet enters the system and moves from the first to the last queue. Yet that is a mistake: it is true that a new packet entering the system modifies the (former) state $(C - 1, C - 1, 0)$ increasing the importance, because a rare event can be generated by having $(q_1 = C \wedge q_2 = C)$; but the importance *decreases* if the packet reaches the third queue, since that queue is a long way from becoming full.

Instead, using the $(\max, +)$ semiring results in the global importance function $Q1+\max(Q2, Q3)$ as previously discussed. So the importance remains unchanged, equal to $2C - 1$ say, while this new packet is in the first or second queue. However, the importance decreases as desired to $2(C - 1)$, say, when the packet moves into the third queue.

4.3.4 Post-processing the functions

There is more one can do once a proper composition strategy has been decided upon. Recall that when a composition operand is selected, Algorithm 4 will choose the proper neutral element, and use it as the *null* function whenever the projection of the DNF expression onto the module is empty. However, when one chooses a ring or a semiring composition strategy there are two operands at play. So for instance when using $(\max, +)$, the *null* function must behave as $\lambda x. 0$ when the (identifier representing the) corresponding local importance function appears as argument of $+$, and it must behave as $\lambda x. -\infty$ when it appears as argument of \max .

One way to achieve the correct behaviour is to apply a *post-processing* to the importance values computed by the derivation algorithms. The $(\max, +)$ semiring is in no real need of such tricks since our importance functions are non-negative. Thus 0 is the null element for both $+$ and \max . The $(+, *)$ ring however could render to zero a whole product in an expression, simply because one of the local importance functions involved is currently at its minimum, viz. 0 according to Algorithm 1. We will refer to this as the *nullification problem*.

The simplest workaround is *value shifting*: take all the values of any local importance function and add 1 to them. Alas, there are no more zeroes and the nullification problem vanishes. All this without the importance function having really changed. Another possibility a little more far fetched is to apply *exponentiation*. Take any base $b \in [1, \infty)$ and change the importance value i to the value b^i for every state. Since $b^0 = 1$ this achieves the goal just

as well; however the importance function has changed, it has expanded and has now an increased range of different (global) importance values.

To see this at work consider the triple tandem queue for the rare event defined by eq. (15). Say the ring $(+, *)$ is used to compose the local importance functions $\{Q1, Q2, Q3\}$, yielding the global function $Q1*(Q2+Q3)$ as described in Section 4.3.3. Since $Q1$ is multiplied by the result of the sum of the other two functions, a post-processing is needed to avoid the nullification problem. For capacity $C = 2$ and assuming $\text{Im}(Q_i) = \{0, 1, 2\}$, the shift post-processing yields *eleven* different global importance values, namely $\{2, 3, 4, 5, 6, 8, 9, 10, 12, 15, 18\}$. Using instead the post-processing `exp` with e.g. base $b = 2.0$ yields $\{2, 3, 4, 5, 6, 8, 10, 12, 16, 20, 24, 32\}$ as global importance values, which are *twelve* in total. If the capacity of the queues is increased to $C = 3$, then shifting yields 19 different global importance values whereas exponentiation yields 22; for $C = 4$ these become 27 vs. 35.

What happens is that many global importance values appear repeated for different combinations of values of the functions $\{Q1, Q2, Q3\}$. This is due to the interactions between product and addition in the expression $Q1*(Q2+Q3)$, which for very close values of $Q1$, $Q2$, and $Q3$ cause many repetitions of the arithmetic result. Setting the possible values of these local functions further apart, e.g. by means of the exponentiation post-processing, results in less such repetitions and thus a richer set of global importance values to choose thresholds from.

The previous section demonstrates the benefits of exploiting the expression of the rare event in order to derive a natural and efficient importance function. In this section we show that a careful choice of post-processing for the specific ring/semiring chosen, can boost the importance range exhibited by the global importance function. The gain is clear: the more importance values the function can yield, the more options to choose thresholds from, and the more promising an application of multilevel splitting becomes.

The gain attained by such techniques will be practically demonstrated when revisiting the database system with redundancy from Example 7. Before doing so however, we will extend the expressiveness of our modelling techniques.

4.4 Input/Output Stochastic Automata

When introducing Algorithms 1 to 4 we highlighted that the Markovian property was never part of the hypotheses, and that all the theory presented

is applicable to general time-homogeneous stochastic processes. Yet the single modelling syntax introduced so far is the PRISM input language, which for the scope of this thesis can only represent Markovian systems. We now present a modern modelling language which can represent processes in a continuous time setting where arbitrary probability distributions—and not just the exponential—can be employed.

Stochastic Automata (SA) were introduced in Section 2.1. They allow sampling stochastic events from general distributions, i.e. arbitrary continuous random variables can be represented in a stochastic automaton. These random variables are denoted *clocks*, and take positive values which result from the sampling of their associated (continuous) distribution. As the global system time advances the *remaining time* of the clocks decreases *synchronously* in equal proportion, viz. the value of all clocks decreases at the same rate. When, as a result of this decrease, the value of a clock becomes zero, *the clock expires* and enables the firing of synchronisation events.

However, the transition relation \rightarrow from Definition 4 of SA allows non-deterministic behaviour, which is problematic from the point of view of simulations. This issue is acknowledged by [DLM16], who derive a subset of SA closed under parallel composition called *Input/Output Stochastic Automata*. The result are fully probabilistic systems, viz. where nondeterminism has been ruled out in the resulting fully composed model.

In order to achieve this goal [DLM16] restrict the framework of SA and work with a partition of the actions set A , splitting them into *input actions* ($A^{\mathcal{I}}$) and *output actions* ($A^{\mathcal{O}}$). Inputs synchronise with outputs, which respectively behave in a reactive and generative manner [vGSS95]. This roughly means that the act of performing the transition (*generating* behaviour) is indicated by an output, whereas inputs listen and synchronise themselves with outputs (*reacting* to this behaviour).

Thus, output actions have an active role and are locally controlled. As a consequence their occurrence time is controlled by a random variable. Instead, input actions have a passive role and are externally controlled. Therefore their occurrence time can only depend on their interaction with outputs. The formal definition of these systems is given next.

Definition 17 (IOSA, [DLM16]). An *Input/Output Stochastic Automaton* (IOSA) is a tuple $(S, A, \mathcal{C}, \rightarrow, s_0, \mathcal{C}_0)$ where:

- S is a denumerable set of states,
- A is a denumerable set of labels partitioned into disjoint sets of input labels $A^{\mathcal{I}}$ and output labels $A^{\mathcal{O}}$,

- \mathcal{C} is a finite set of clocks s.t. each $x \in \mathcal{C}$ has an associated continuous probability measure $\mu_x: \mathbb{R} \rightarrow [0, 1]$ with support on $\mathbb{R}_{>0}$,
- $\rightarrow \subseteq S \times 2^{\mathcal{C}} \times A \times 2^{\mathcal{C}} \times S$ is a transition function,
- $s_0 \in S$ is the initial state, and
- $\mathcal{C}_0 \subseteq \mathcal{C}$ are the clocks initialised in the initial state.

In addition to the above, an IOSA satisfies the following constraints:

- if $s \xrightarrow{C,a,C'} s'$ and $a \in A^{\mathcal{I}}$, then $C = \emptyset$,
- if $s \xrightarrow{C,a,C'} s'$ and $a \in A^{\mathcal{O}}$, then C is a singleton set,
- if $s \xrightarrow{\{x\},a_1,C_1} s_1$ and $s \xrightarrow{\{x\},a_2,C_2} s_2$, then $a_1 = a_2$, $C_1 = C_2$, and $s_1 = s_2$,
- if $s \xrightarrow{\{x\},a,C} s'$ then, for every transition $t \xrightarrow{C_1,b,C_2} s$, either $x \in C_2$, or $x \notin C_1$ and there exists a transition $t \xrightarrow{\{x\},c,C_3} t'$,
- if $s_0 \xrightarrow{\{x\},a,C} s$ then $x \in \mathcal{C}_0$,
- for every $a \in A^{\mathcal{I}}$ and state s , there exists a transition $s \xrightarrow{\emptyset,a,C} s'$,
- for every $a \in A^{\mathcal{I}}$, if $s \xrightarrow{\emptyset,a,C_1} s_1$ and $s \xrightarrow{\emptyset,a,C_2} s_2$, then $C_1 = C_2$ and $s_1 = s_2$.

The occurrence of an action is controlled by the expiration of clocks. Thus, whenever $s \xrightarrow{\{x\},a,C} s'$ and the system is in state s , output action a will occur as soon as clock x expires. At this point the system moves to state s' , choosing new values for every clock $y \in C$ sampled from the corresponding distribution μ_y . For input transitions $s \xrightarrow{\emptyset,a,C} s'$ the behaviour is similar; the difference lies in the time of occurrence of the transition, which will be defined when the action interacts with an output.

Constraint (a) states that inputs are reactive and hence their occurrence is controlled by the environment. Constraint (b) states that outputs are generative (or locally controlled) so they have an associated set of clocks which determine their occurrence time[†].

Constraint (c) forbids that a single clock enables two different transitions, which is crucial to avoid nondeterministic behaviour, since otherwise two output actions could become enabled simultaneously. Furthermore notice that using a clock after it has expired would immediately enable the respective output transition. That also leads to situations where two or more

[†] The set is a singleton in the wake of achieving a clean definition.

transitions are simultaneously enabled, e.g. if the system arrives at a state where two different (expired) clocks enable two different output transitions. Constraints (d) and (e) ensure that expired clocks are not used. Particularly (d) states that an enabling clock x at state s must either: be set on arrival to state s ($x \in C_2$); or is enabling in the immediately preceding state t and it has not been used right before reaching s ($x \notin C_1$).

Since clock values are sampled from continuous random variables, the probability that the value sampled for two different clocks coincides is zero. This, together with constraints (c) to (e), guarantees that *almost never* two different output transitions are enabled at the same time point. Finally, constraints (f) and (g) are usual restrictions on Input/Output-like automata: (f) ensures that outputs are not blocked in a composition; (g) ensures that determinism is preserved by the parallel composition.

Input/Output Stochastic Automata are given semantics over NLMP [Wol12, DSW12], which are a generalisation of probabilistic transition systems with continuous domain. More precisely, NLMP extend LMP [DEP02] with *internal* nondeterminism. We next define NLMP formally, since they will be used to show that IOSA are deterministic. For a deeper understanding of Definition 18 and a more extensive description of these systems and its properties, we refer the interested reader to Appendix C.

Definition 18 (NLMP). A *nondeterministic labelled Markov process* (NLMP) is a tuple $(\mathbf{S}, \Sigma, \{\mathcal{T}_a \mid a \in \mathcal{L}\})$ where:

- \mathbf{S} is an arbitrary set of states,
- Σ is a σ -algebra on \mathbf{S} ,
- for each label $a \in \mathcal{L}$ the function $\mathcal{T}_a: \mathbf{S} \rightarrow \Delta(\Sigma)$ is measurable from Σ to the hit σ -algebra $H(\Delta(\Sigma))$.

The semantics of an IOSA is formally defined by an NLMP using two types of transitions: one type encodes the discrete steps, and contains all probabilistic information introduced by the sampling of clocks; the other type describes the time steps, recording the passage of time by synchronously decreasing the value of all clocks. To simplify matters, Definition 19 taken from [DLM16] assumes an order in the set of clocks \mathcal{C} , affecting also the vectors in \mathbb{R}^N representing their valuations.

Definition 19. Given an IOSA $\mathcal{I} = (S, A, \mathcal{C}, \rightarrow, s_0, \mathcal{C}_0)$ with $\mathcal{C} = \{x_i\}_{i=1}^N$, its semantics is defined by the NLMP $\mathcal{P}(\mathcal{I}) = (\mathbf{S}, \mathcal{B}(\mathbf{S}), \{\mathcal{T}_a \mid a \in \mathcal{L}\})$ where:

- $\mathbf{S} = (S \uplus \{\text{init}\}) \times \mathbb{R}^N$ and $\mathcal{L} = \{\text{init}\} \uplus A \uplus \mathbb{R}_{>0}$, with $\text{init} \notin S \cup A \cup \mathbb{R}_{>0}$,
- $\mathcal{T}_{\text{init}}(\text{init}, \vec{v}) = \{\delta_{s_0} \times \prod_{i=1}^N \mu_{x_i}\}$,
- $\mathcal{T}_a(s, \vec{v}) = \{\mu_{\vec{v}, C', s'} \mid s \xrightarrow{C, a, C'} s', \bigwedge_{x_i \in C} \vec{v}(i) \leq 0\}$ for all $a \in A$, where $\mu_{\vec{v}, C', s'} = \delta_{s'} \times \prod_{i=1}^N \bar{\mu}_{x_i}$, with $\bar{\mu}_{x_i} = \mu_{x_i}$ if $x_i \in C'$ and $\bar{\mu}_{x_i} = \delta_{\vec{v}(i)}$ otherwise, and
- $\mathcal{T}_d(s, \vec{v}) = \{\delta_{(s, \vec{v})}^{-d} \mid 0 < d \leq \min(V)\}$ for all $d \in \mathbb{R}_{\geq 0}$, where $\delta_{(s, \vec{v})}^{-d}$ is the Dirac distribution $\delta_s \times \prod_{i=1}^N \delta_{\vec{v}(i)-d}$, and we define the set of positive reals $V = \left\{ \vec{v}(i) \mid \exists a \in A^{\mathcal{O}}, C' \subseteq C, s' \in S : s \xrightarrow{\{x_i\}, a, C'} s' \right\}$, with $\min(\emptyset) \doteq \infty$.

The fact that $\mathcal{P}(\mathcal{I})$ from Definition 19 actually satisfies Definition 18 of NLMP is proved in [DLM16]. Notice the state space \mathbf{S} of $\mathcal{P}(\mathcal{I})$ is the product space of the states of the IOSA with all possible clock valuations. A distinguished initial state init is added to encode the random initialization of all clocks. In turn, \mathbf{S} has the usual Borel σ -algebra structure, $\mathcal{B}(\mathbf{S})$.

Discrete steps are encoded by \mathcal{T}_a for $a \in A$. At state (s, \vec{v}) the transition $s \xrightarrow{C, a, C'} s'$ takes place if $\bigwedge_{x_i \in C} \vec{v}(i) \leq 0$, i.e. once all current clocks have expired—trivially true for input actions. The next state reached in the NLMP will have s' as IOSA state, clocks not in C' preserve their values, and clocks in C' have their values resampled from their respective distributions.

Time steps are encoded by $\mathcal{T}_d(s, \vec{v})$ for $d \in \mathbb{R}_{\geq 0}$. Such transition can only take place iff there is no output transition enabled in the current state within the next d time units. If that is actually the case then the system remains in the same IOSA state s , and all clock values are decreased by d , viz. d units of time are spent on state s .

The IOSA modelling formalism was designed to allow the parallel composition of several system components. Owing to its input/output foundations, output actions are autonomous and can only synchronise with homonymous input actions; in other words, synchronization between output actions of different components is not allowed. Further technical aspects need to be accounted for prior to defining a synchronisation mechanism, like name clashing of the clocks. The following definitions taken from [DLM16] formalise the notion of parallel composition for IOSA.

Definition 20. Given two IOSA $\mathcal{I}_1 = (S_1, A_1, C_1, \rightarrow_1, s_0^1, C_0^1)$ and $\mathcal{I}_2 = (S_2, A_2, C_2, \rightarrow_2, s_0^2, C_0^2)$, these are *compatible* if they do not share output actions nor clocks, that is, if $A_1^{\mathcal{O}} \cap A_2^{\mathcal{O}} = \emptyset$ and $C_1 \cap C_2 = \emptyset$.

Definition 21 (IOSA composition). Given two compatible IOSA \mathcal{I}_1 and \mathcal{I}_2 , its *parallel composition* $\mathcal{I}_1 \parallel \mathcal{I}_2$ is a tuple $(S, A, \mathcal{C}, \rightarrow, (s_0^1, s_0^2), \mathcal{C}_0)$ where:

- $S = S_1 \times S_2$,
- $A = A^{\mathcal{O}} \uplus A^{\mathcal{I}}$ s.t. $A^{\mathcal{O}} = A_1^{\mathcal{O}} \uplus A_2^{\mathcal{O}}$ and $A^{\mathcal{I}} = (A_1^{\mathcal{I}} \uplus A_2^{\mathcal{I}}) \setminus A^{\mathcal{O}}$,
- $\mathcal{C} = \mathcal{C}_1 \uplus \mathcal{C}_2$,
- $\mathcal{C}_0 = \mathcal{C}_0^1 \uplus \mathcal{C}_0^2$,
- \rightarrow is the smallest relation defined by the following rules:

$$\frac{s_1 \xrightarrow{C, a, C'}_1 s'_1}{(s_1, s_2) \xrightarrow{C, a, C'} (s'_1, s_2)} \quad a \in A_1 \setminus A_2 \qquad \frac{s_2 \xrightarrow{C, a, C'}_2 s'_2}{(s_1, s_2) \xrightarrow{C, a, C'} (s_1, s'_2)} \quad a \in A_2 \setminus A_1$$

$$\frac{s_1 \xrightarrow{C_1, a, C'_1}_1 s'_1 \quad s_2 \xrightarrow{C_2, a, C'_2}_2 s'_2}{(s_1, s_2) \xrightarrow{C_1 \cup C_2, a, C'_1 \cup C'_2} (s'_1, s'_2)}$$

Definition 21 provides structural rules to build the (*syntactic*) parallel composition of two compatible IOSA, but it does not give any insight on whether the resulting tuple is itself an Input/Output Stochastic Automaton. For that purpose [DLM16] show that the constraints from Definition 17 of IOSA are also satisfied by $\mathcal{I}_1 \parallel \mathcal{I}_2$.

Theorem 8 (IOSA are closed over parallel composition, [DLM16]). *Let \mathcal{I}_1 and \mathcal{I}_2 be two compatible IOSA. Then $\mathcal{I}_1 \parallel \mathcal{I}_2$ is also an IOSA.*

A *closed IOSA* is an Input/Output Stochastic Automaton resulting from the parallel composition of two or more IOSA, where all synchronizations have been resolved. Definition 21 thus ensures that a closed IOSA will have no input actions, i.e. $A^{\mathcal{I}} = \emptyset$.

[DLM16] shows that closed IOSA are deterministic, which makes them amenable to analysis by discrete event simulation. An IOSA is *deterministic* if (almost surely) at most one discrete transition is enabled at every time point. Equivalently, [DLM16] call deterministic an IOSA which *almost never* reaches a state where two different discrete transitions are enabled. The formalisation of this concept, which is given next, requires resorting to the NLMP semantics of the automaton.

Definition 22 (Deterministic IOSA). An IOSA \mathcal{I} is *deterministic* whenever in $\mathcal{P}(\mathcal{I}) = (\mathcal{S}, \mathcal{B}(\mathcal{S}), \{\mathcal{T}_a \mid a \in \mathcal{L}\})$, a state $(s, \vec{v}) \in \mathcal{S}$ such that the set

$\bigcup_{a \in AU\{\text{init}\}} \mathcal{T}_a(s, \vec{v})$ contains more than one probability measure, is almost never reached from any initial state $(\text{init}, \vec{v}') \in \mathcal{S}$.

By *almost never* [DLM16] mean that the measure of the set of paths leading to a state $(s, \vec{v}) \in \mathcal{S}$ where $1 < |\bigcup_{a \in AU\{\text{init}\}} \mathcal{T}_a(s, \vec{v})|$, is zero. Moreover, Definition 22 requires the NLMP $\mathcal{P}(\mathcal{I})$ to satisfy the notions of *time additivity*, *time determinism*, and *maximal progress* [Yi90]; all this is proved to hold in [DLM16]. In particular, maximal progress means that whenever an output transition is enabled, time cannot advance in a state, but rather the output shall be performed first.

So far we have only described the nature and properties of the IOSA modelling formalism. To apply the splitting simulation techniques from this thesis to such formalism we would require to work exclusively with systems satisfying Definition 22. Happily, [DLM16] show that once all synchronizations have been resolved, the resulting *fully composed IOSA* is indeed deterministic as per Definition 22.

Theorem 9 ([DLM16]). *Every closed IOSA is deterministic.*

This chapter is devoted to developing techniques which exploit the compositional nature of a system model. Theorem 9 enables us to choose Input/Output Stochastic Automata as the modelling formalism with which we will verify the efficiency of the methods introduced so far.

4.5 Automation and tool support

The overall theory and strategies supporting our compositional approach to importance splitting has been covered in sections 4.1 to 4.4. This section studies some practical aspects, aiming at the development of software tools to implement such approach.

4.5.1 Selection of the thresholds

In the case studies presented along Section 3.5, the thresholds selection mechanism was the object of many critiques. This was rather disappointing since the algorithm implemented in the BLUEMOON tool, named Adaptive Multilevel Splitting, has the advantage of *dynamically moulding* the selection to the particular system under study.

Our implementation of Adaptive Multilevel Splitting was shown in Algorithm 2. It runs pilot simulations on the system model \mathcal{M} , whose statistical evaluation w.r.t. the importance values observed yielded the threshold importance values used later by RESTART. From the theoretical viewpoint, this complemented nicely the static analysis of the system and of the rare event used by Algorithm 1 to derive the importance function.

Its adaptability notwithstanding, Algorithm 2 proved to be quite sensitive to the global splitting value selected, and even to the particular simulation run, characterised by the seed fed to the Random Number Generator. In occasions re-running the experiment produced better results, viz. faster convergence, related to a different choice of thresholds—see e.g. Section 3.5.3.

This suggests the statistical properties of the algorithm are not optimal. In that respect recall that the subroutine $\mathcal{M}.\text{simulate_ams}(s, n, m, f, \text{sim})$ is called once per iteration of the main loop, in order to select a threshold with higher importance than the one previously selected. That routine launches n simulations from state s with predetermined lifetime $m \in \mathbb{R}_{>0}$. The outcomes of these simulations, in terms of importance values measured by the function f , determines the next threshold.

Notice that making all n simulations start from the same state s introduces a potentially high correlation in the outcomes of the runs. This is recognised by [CDMFG12], who have developed a much more sound algorithm (from the statistical point of view) named *Sequential Monte Carlo*.

The main difference between Sequential Monte Carlo and its predecessor Adaptive Multilevel Splitting lies in the selection of the starting states at each iteration of the main loop. With the idea of reducing the correlation between the resulting runs, Sequential Monte Carlo chooses these states independently from among all the states in the system. The only condition they must comply to is having an importance value assigned by function f greater than the last selected threshold.

In [CDMFG12] the states are particles generated from a Markov kernel. Thus, drawing n independent and identically distributed new particles to start simulations from, is only a matter of resampling from the kernel. In contrast, from our simulation perspective on IOSA models, it makes more sense to choose only among *reachable states*. Besides, since our scenario is discrete, we can afford to choose states to which function f gives *exactly* the importance value chosen as the previous threshold.

The simplified pseudocode of our implementation of Sequential Monte Carlo is presented in Algorithm 5. We highlight that both the input and the output are the same than for our implementation of Adaptive Multi-

level Splitting, viz. Algorithm 2. The only substantial difference between both algorithms is that subroutine `choose_dist(...)` in Algorithm 5 is used to select the starting states at each iteration of the main loop, and that `simulate_smc(...)` starts from n potentially different states, instead of starting from a single state like `simulate_ams(...)` does in Algorithm 2.

Particularly, $\mathcal{M}.\text{choose_dist}(\dots)$ takes the array of states $\mathbf{sim} \in S^{n+k}$ as one of its inputs. The general idea is to use the first n positions of \mathbf{sim} to find the new thresholds, storing there the states resulting from pilot runs launched for that purpose. In contrast, the last k positions of \mathbf{sim} hold states with the same importance as the last threshold found. At each step, $\mathcal{M}.\text{choose_dist}(\dots)$ launches n pilot runs; these will start from states randomly sampled from the last k positions of \mathbf{sim} , i.e. from a random sample of the last threshold found. The resulting states of those n simulations, i.e. those which achieved maximum importance, will be stored in the first n positions of \mathbf{sim} , to check whether a new (higher) threshold has been found.

Even more in detail, the subroutine $\mathcal{M}.\text{choose_dist}(\mathbf{sim}, n, k, t, f)$ performs n independent simulations, which run until a state to which $f: S \rightarrow \mathbb{N}$ assigns importance $t \in \mathbb{N}$ is found. The starting states for these simulations are randomly chosen from the states at position $n, n+1, \dots, n+k-1$ of \mathbf{sim} . Upon reaching a state with importance equal to t , each of the n simulations stops and saves such state in the corresponding i -th position of \mathbf{sim} , for $i \in \{0, 1, \dots, n-1\}$. Finally, k states from among those n states are randomly selected and copied into positions $n, n+1, \dots, n+k-1$ of \mathbf{sim} , to be used as initial states in the next invocation of `choose_dist(...)`.

Three more remarks will be useful to better grasp Algorithm 5:

1. `sort(sim, f, i, n)` sorts the states of the array \mathbf{sim} which are in positions $i, i+1, \dots, i+n-1$, in increasing order according to the values that function $f: S \rightarrow \mathbb{N}$ assigns them;
2. $\mathcal{M}.\text{simulate_smc}(\mathbf{sim}, n, m, f)$ operates like `simulate_ams(...)` from Algorithm 2, only it starts the n simulations from the states at positions $0, 1, \dots, n-1$ of the array of states \mathbf{sim} , and leaves in those positions the states resulting from such simulations;
3. when an iteration cannot find a new threshold, the main loop is broken (instruction **break loop**) and we fall back on `choose_remaining(T, f)` which, observing that $T.\text{back}() < \max(f)$, chooses thresholds between $T.\text{back}()$ and $\max(f)$ following some heuristic guaranteed to terminate.

number of steps that each simulation launched by `choose_dist(...)` can perform, yields finite termination for that subroutine. Finally and by hypothesis, `choose_remaining(...)` terminates after executing a finite number of instructions. \square

4.5.2 IOSA model syntax

Proposition 10 above speaks of simulations run on finite IOSA. However, this modelling formalism has been presented in Section 4.4 from a purely theoretic perspective, just as it was developed by [DLM16]. To choose IOSA as the language in which our system models are to be expressed, we need a concrete syntax whose grammar shall produce automata complying to Definitions 17 and 22.

To that aim we have developed the following syntax[†], which we will present informally as we did with the PRISM input language in Section 3.3.1. The constructs of this *IOSA model syntax* are, as a matter of fact, quite similar to those of PRISM, with the major addition of variables of type *clock* whose values must be sampled from stochastic distributions. We next provide an exhaustive list of differences between the PRISM input language as used in this thesis, and the IOSA model syntax to be used for experimentation in the sections to come:

- at global scope only constants, properties, and modules can be defined;
- constants must be of either Boolean, integral, or floating point type;
- properties can be specified either in a dedicated file, or in the model file enclosed in a `properties...endproperties` environment;
- property queries are specified one per line and are either of type
 - ▷ *transient*, following the format `P(!stop U rare)`, or
 - ▷ *steady-state*, following the format `S(rare)`,

where `stop` and `rare` are Boolean-valued expressions representing the stopping and rare event conditions respectively;

- variables can only appear within a module body, viz. enclosed in a `module...endmodule` environment;

[†] My colleague Raúl E. Monti and my advisor Pedro R. D'Argenio were majorly in charge of this task; credit should go to them.

- variables must be of either Boolean, (ranged) integral, or clock type;
- each clock variable must be mapped to exactly one continuous probability function, and can only be assigned randomly chosen values, resulting from a sampling of such function;
- non-empty labels in the edges of a module must be decorated either with ? to signify that it is an input action (and thus an *input edge*), or with ! to signify that it is an output action (resp. *output edge*);
- an empty label indicates a non-synchronizing output edge;
- an empty Boolean guard in an edge is interpreted as `true`;
- a semicolon immediately following symbol `->` is interpreted as NOP;
- besides a Boolean guard, output edges must declare one clock name between the character `@` and the symbol `->`, which links that clock variable to the concrete output transitions represented by the edge.

To show how this syntax looks like, we present in Code 4.3 an extract of a model described using the IOSA model syntax. The system represented is the modularised tandem queue introduced in Code 3.3. Incidentally, since all clock variables are mapped to the exponential distribution, the resulting IOSA model is equivalent to a CTMC, as expected.

Code 4.3: IOSA model for the tandem queue (extract)

```

1  const int c = 8; // Capacity of both queues
    :
14 module Arrivals
15     clk0: clock; // External arrivals ~ Exponential(lambda)
16     [P0!] @ clk0 -> (clk0' = exponential(lambda));
17 endmodule
18
19 module Queue1
20     q1: [0..c];
21     clk1: clock; // Queue1 processing ~ Exponential(mu1)
22     // Packet arrival
23     [P0?] q1 == 0      -> (q1' = q1+1) & (clk1' = exponential(mu1));
24     [P0?] q1 > 0 & q1 < c -> (q1' = q1+1);
25     [P0?] q1 == c      -> ;
26     // Packet processing
27     [P1!] q1 == 1 @ clk1 -> (q1' = q1-1);
28     [P1!] q1 > 1 @ clk1 -> (q1' = q1-1) & (clk1' = exponential(mu1));

```

```

29 endmodule
    :
43 properties
44     P( q2 > 0 U q2 == c ) // transient
45     S( q2 == c ) // steady-state
46 endproperties

```

Notice that the single edge of module `Arrivals` in line 16 of Code 4.3, is an output edge since its action label `P0` is decorated with `!`. Its Boolean guard (between characters `]` and `@`) is empty and thus equivalent to `true`. Moreover, this output edge is associated to the clock `clk0`, which is mapped to an exponential probability density function of rate `lambda`.

The output action `P0` from module `Arrivals` synchronises with the input action `P0` from module `Queue1`, i.e. with any of the edges in lines 23–25. The Boolean guards of those edges form a partition of the range of the integral variable `q1`. Therefore, the output edge of module `Arrivals` is always enabled *from a logical point of view*, and will synchronise with exactly one of the input edges in lines 23–25. *From a temporal point of view* and by definition of IOSA, the output edge becomes enabled when clock `clk0` expires.

This synchronisation mechanism resembles that of the PRISM model for the tandem queue from Code 3.3. Such coincidence is to be expected, since the IOSA model syntax was originally inspired in the PRISM input language.

Notice that the effects of an edge, i.e. the consequences of taking the edge which are described after the symbol `->`, appear enclosed in parentheses and concatenated with the character `&`. For instance the input edge in line 23 has two effects: incrementing by one the value of the integral variable `q1`, and assigning a fresh random value to the clock variable `clk1`, sampled from an exponentially distributed probability density function of rate `mu1`.

Notice in particular the input edge in line 25 of Code 4.3. This edge has an empty effect, viz. a semicolon appears immediately following `->`. This is interpreted as a NOP or SKIP, that is, the absence of an effect. Semantically that edge represents a packet trying to enter a fully occupied first queue, which is promptly discarded.

In Code 4.3 the property queries are included in the model file, and hence appear enclosed in a `properties...endproperties` environment. The property in line 44 is transient: it asks the probability of observing a saturated second queue before the queue empties. The property in line 45 is steady-state: it asks the time proportion that the second queue spends in a saturated state, viz. the long run probability of a saturation in that queue.

4.5.3 Finite Improbability Generator

We have developed the software tool FIG, which implements the compositional approach to multilevel splitting described in this chapter. It is written in pure C++ and is standalone software. The full name of the tool is *Finite Improbability Generator*, owing to Douglas Adam's masterpiece. FIG is freely available at <http://dsg.famaf.unc.edu.ar/tools> under the terms of the General Public License (GPL v3).

The inputs, that is the model file and property queries, must be specified following the IOSA model syntax described in Section 4.5.2. From now on we will assume that the property queries are specified within the file where the model is described, as in Code 4.3.

Remarkably, FIG also supports (certain types of) models described in the *jani model specification format* version 1.0 (JANI, [BDH⁺17]). On the one hand the IOSA formalism subsumes CTMC, so continuous-time Markov chains described in JANI can be read-in by the tool. On the other hand Stochastic Timed Automata (STA) subsume IOSA, and FIG can operate with certain STA models described in JANI. Specifically, deterministic STA complying with Definitions 17 and 22 can be accepted by the tool.

The most basic invocation of FIG requires three mandatory options: `<model>`, `<termination>`, and `<strategy>`. Their syntax and semantics can be briefly described as follows:

`<model>` The path to the file with the IOSA (or JANI) model description and property queries.

`<termination>` The stopping criterion (or criteria). It can be:

- `--stop-conf` to simulate until the specified confidence coefficient and relative precision are achieved. Those two parameters must be numbers in the open interval $(0, 1)$;
- `--stop-time` to simulate for the specified amount of (wall-clock) time, described in the format `<digit>+[<s/m/h/d>]`.

`<strategy>` The strategy used to simulate. It must be one of:

- `--flat` to perform standard Monte Carlo simulations;
- `--amono` to perform RESTART simulations using an automatic monolithic importance function, built using the approach from Chapter 3;

- `--acomp` to perform RESTART simulations using an automatic compositional importance function, built using the approach from this chapter and a composition operand (or strategy) given as mandatory parameter;
- `--adhoc` to perform RESTART simulations using an ad hoc importance function specified by the user as mandatory parameter.

The order of the options is arbitrary. For instance the line

```
>_ fig model.sa --flat --stop-conf .9 .4
```

invokes the tool to perform standard Monte Carlo simulations on the IOSA file `model.sa`. For each of the property queries specified within the file, simulations will be launched until a confidence interval of 90% confidence level and 40% relative precision (i.e. a relative error of 20%—see Definition 6) is built around the estimated value of the property.

Both `<model>` and `<strategy>` are simple options, whereas `<termination>` is a multi-option. This means several stopping criteria can be specified, and independent estimations are run to meet each of them. For instance

```
>_ fig model.sa --amono --stop-time 5m --stop-conf .9 .4
```

launches, for each property query specified in `model.sa`: first an estimation lasting 5 minutes of (wall time) execution, for which typical confidence intervals around the estimate are reported; and then an estimation which will run until an interval of 90% confidence level and 40% relative precision is built around the estimate.

Since the `--amono` option was specified in the command above, all those simulations will use multilevel splitting. Like BLUEMOON, FIG implements RESTART to perform splitting simulations. The `--amono` option makes FIG build an (*automatic*) monolithic importance function for each property, subsequently used in the RESTART simulations.

Unlike BLUEMOON, the FIG tool does not require to be told which kind of simulation (e.g. transient vs. steady-state) to run for each property query. This is deduced from the logical expression: for transient properties several independent simulations are launched, whose average yields the desired estimate; for steady-state properties the batch means method is employed.

As earlier stated, choosing the `--adhoc` strategy requires the user to provide as parameter the importance function he desires to use. Such function

must have image on the natural numbers, and any constant or variable from the system model can appear in the arithmetic expression that defines it. For instance if `tandem_queue.sa` contains the IOSA model of the tandem queue from which Code 4.3 was extracted, then

```
>_ fig tandem_queue.sa --ad hoc "q1+5*q2" --stop-conf .9 .4
```

will run RESTART simulations to estimate the value of both properties (recall these were $P(q_2 > 0 \cup q_2 = c)$ and $S(q_2 = c)$), using the ad hoc importance function which adds five times the number of packets in the second queue to the number of packets in the first queue, i.e. $q_1 + 5 * q_2$.

The situation is quite different for the `--acom` option, although a parameter is also mandatory. A composition operand can be chosen, for example

```
>_ fig tandem_queue.sa --acom "+" --stop-conf .9 .4
```

invokes the tool to perform estimations like in the situation described above, but using an (*automatic*) compositional importance function for the RESTART simulations. Addition is specified as composition operand, meaning the global importance function used will be $A + Q_1 + Q_2$, where A , Q_1 , and Q_2 stand for the local importance functions built for modules `Arrivals`, `Queue1`, and `Queue2` respectively. Recall these functions are built anew for each property query, since the definition of the rare event could differ from one query to the next.

Rather than a composition operand, the `--acom` option can also take an ad hoc composition strategy, defined by the user in the way described in Section 4.3.1. It is worthy to mention that unlike the `--ad hoc` option, the arithmetic expression passed as parameter to `--acom` must contain names of modules of the system, which are interpreted as the local importance functions built for such modules. This means that the command

```
>_ fig tandem_queue.sa --acom "Arrivals+Queue1+Queue2" \
  --stop-conf .9 .4
```

imitates the previous invocation, which used addition as composition operand.

Much more importantly, this facility allows a straightforward implementation of the ring/semiring strategy from Section 4.3.3. Suppose e.g. the triple tandem queue modelled with the PRISM input language in Code 4.2, is described with the IOSA model syntax in the file `3tandem_queue.sa`. Assuming the same names are given to the modules, the command

```
>_ fig 3tandem_queue.sa \
    --acomp "Queue1+max(Queue2,Queue3)" --stop-time 2h
```

uses the semiring (max, +) as described in Section 4.3.3 to compose the local importance functions built for the modules of the queues.

So far the importance function specification has been discussed, but multilevel splitting simulations (i.e. whenever `--adhoc`, `--amono`, or `--acomp` are chosen) also require selecting the thresholds prior to running RESTART. To that aim and by default FIG runs Algorithm 5, using the importance function built for the current property query. This can be changed by means of the option `--thresholds` to use Adaptive Multilevel Splitting, a fixed (e.g. non-adaptive) strategy, or pure Sequential Monte Carlo. Algorithm 5 can be described as implementing a *hybrid strategy*: upon a failure of Sequential Monte Carlo the subroutine `choose_remaining(...)` is invoked, which implements a non-adaptive strategy guaranteed to terminate.

FIG can take several additional options to customise the estimation procedures. The full list together with their invocation syntax and some practical examples is obtained with the `--help` option. We next describe the most relevant among these options, two of which were used to run the experiments reported in Section 4.6. Also and from here onward, estimations whose termination is specified using the `--stop-conf` option will be referred to as *confidence-bound estimations* or merely *confidence estimations*. Analogously *time-bound estimations* or simply *time estimations* will refer to estimations whose termination is specified by means of the `--stop-time` option.

Regardless of the particular stopping criteria chosen, a maximum wall-clock execution time can be imposed by means of the option `--timeout`, which takes the same mandatory parameter than the `--stop-time` option. The `--timeout` option is very useful for confidence-bound estimations, when one has no idea whatsoever how long the estimation could take. Instead for time-bound estimations, if the `--timeout` option is given then simulations will last for the shortest of the two time lapses.

We emphasise that the layout of the output differs between time (or timed-out) estimations and confidence estimations. If the estimation finished upon reaching the desired confidence level and relative precision for the interval, the final outcome looks as follows:

```
>_ ~~~~~
    · FIG ·
    ~~~~~
```

```

This is the Finite Improbability Generator.
Version: 1.1
Build:   Release
      :
RNG algorithm used: pcg32
Estimating P( (q2>0) U (q2==8) ),
using simulation engine "restart"
with importance function "concrete_coupled"
built using strategy   "auto"
with post-processing   "(null)"
and thresholds technique "hyb"
[ 2 thresholds | splitting 5 ]
Confidence level: 80%
Precision: 40%
RNG seed: 1944391357620130122 (randomized)
· Computed estimate: 5.34e-06 (7344384 samples)
· Computed precision: 1.67e-06
· Precision: 2.13e-06
· Confidence interval: [ 4.27e-06, 6.40e-06 ]
· Estimation time: 29.04 s

```

Notice there is a “Computed precision” and a plain “Precision.” The former is the empiric interval width achieved using the techniques from Section 3.3.4. The latter is the (*theoretic*) relative precision requested, which in this case equals $5.34e-06 \times 0.4 = 2.13e-06$.

Alternatively, estimations could stop due to timing reasons, in which case there is no theoretic precision to report because the theory from Section 3.3.4 cannot be applied. In such cases a set of confidence intervals is displayed, built with the gathered data for typical confidence levels, e.g.

```

>_
      :
[ 2 thresholds | splitting 5 ]
Confidence level: 80%
Precision: 40%
Timeout: 00:00:10
RNG seed: 17463016430344695793 (randomized)
· Computed estimate: 6.54e-06 (2508288 samples)
· 80% confidence
  - precision: 3.00e-06
  - interval: [ 5.04e-06, 8.04e-06]

```



```

· 90% confidence
  - precision: 3.86e-06
  - interval: [ 4.61e-06, 8.47e-06]
· 95% confidence
  - precision: 4.59e-06
  - interval: [ 4.24e-06, 8.84e-06]
· 99% confidence
  - precision: 6.04e-06
  - interval: [ 3.52e-06, 9.56e-06]
· Estimation time: 10.00 s

```

It is important to remark that according to Algorithm 4, functions built automatically use zero as minimum importance value. This means that the local initial state of a module is assigned the value 0 by its local importance function, which can be problematic for the compositional approach if the product is used either as composition operand or in a composition strategy.

Take for instance the command

```
>_ fig tandem_queue.sa --acomp "*" --stop-conf .8 .6
```

which specifies the product to be used as composition operand for the compositional approach. Say the rare event is a saturation in both queues. Then the global importance function will yield the value 0 whenever the first queue is in its initial local state, regardless of the occupancy in the second queue. This is clearly at odds with the desired behaviour. We have referred to this issue in Section 4.3.4 as the nullification problem.

Mostly because of this inconvenience and in line with Section 4.3.4, FIG offers a `--post-process` option to modify the importance of the states once the importance functions have been computed. The (local) importance values of the states can therefore be increased or exponentiated, solving the problem caused by zero being the absorbing element of `*`.

Revisiting the previous situation, the command

```
>_ fig tandem_queue.sa --acomp "*" --stop-conf .8 .6 \
  --post-process shift 1
```

increases by one the local importance values in all the system modules. That means that the lowest value returned by a local importance function is 1 rather than 0. Therefore the nullification problem is solved, since e.g. a first queue situated in its initial state will have local importance 1.

More in detail, the `--post-process` option takes two parameters:

- `<type>` the kind of post-processing to apply, which can be either `shift` to increase/decrease the importance computed for each state by some value, or `exp` to apply exponentiation using as exponent the importance of the state;
- `<arg>` the numeric argument to use, which for `shift` can be any integral constant (the amount to increase/decrease each value by), and for `exp` must be a floating point number greater than 1.0 (the base to use during exponentiation).

To illustrate the use of exponentiation consider the command

```
>_ fig tandem_queue.sa --acomp "*" --stop-conf .8 .6 \
    --post-process exp 2.0
```

This means that in every module, the importance value i of each local state will be replaced by the value 2^i . Again this suffices to solve the nullification problem, since $2^i > 0$ for all $i \in \mathbb{N}$ and in particular $2^0 = 1$, viz. the lowest importance of any queue will be the neutral element of $*$.

This section concludes discussing the interaction of the tool with the JANI model specification format by [BDH⁺17]. FIG can operate as bidirectional translator between the IOSA and JANI formalisms, for which the options `--to-jani` and `--from-jani` are offered. When any of these options is specified the tool assumes the translation role and refrains from running estimations. Nonetheless FIG can be fed a file containing an (IOSA-compatible) JANI model; in such case, upon a successful implicit translation, estimations will be carried out as usual. By means of example let `tandem_queue.jani` be a file with some JANI model corresponding to the modularised tandem queue discussed so far. Then the command

```
>_ fig tandem_queue.jani --amono --stop-conf .95 .6
```

invokes the tool to perform the usual RESTART simulations employing the monolithic importance function, and estimations will finish once an interval of 95% confidence level and 30% relative error is achieved. The translation from the JANI format of the file to the IOSA syntax compiled by FIG is transparent to the user.

Regarding the translation and as discussed, only CTMC and certain type of deterministic Stochastic Timed Automata (STA) comply with the

IOSA formalism. When exporting to JANI with the `--to-jani` option an IOSA-compatible STA is generated. When importing from JANI with the option `--from-jani` several checks are due. The most basic ones comprise the absence of global variables and employing the broadcast-like synchronisation of IOSA. Any CTMC model complying to that should be accepted. STA are more involved since they allow several clock manipulations unknown to Stochastic Automata in general (e.g. setting a deterministic time value in a clock variable), and they also allow several enabling clocks per edge. Hence the constraints from Definition 17 must be thoroughly revised when importing from an STA JANI model:

1. first, constraints (a) and (b) are syntactically checked,
2. then a tentative IOSA model is built,
3. and finally constraints (c) to (g) are evaluated.

Upon success, i.e. if the resulting model described in the IOSA model syntax complies with Definition 17, the file containing the translated model is output. Otherwise an error message is displayed and translation aborts.

4.6 Case studies

Several systems were taken from the RES literature and analysed with FIG. The general description of these systems and the results from experimentation are shown in this section. The Input/Output Stochastic Automata used for such purpose are listed in Appendix A.

4.6.1 Experimentation setting

All models studied here are described in the IOSA model syntax. Some of the Markovian case studies analysed in Section 3.5 are revisited in this section. These tests served to validate the correct functioning of the FIG tool. Non-Markovian systems, which use clocks associated to e.g. the log-normal distribution, are also studied in this section.

Following the general format of Section 3.5 we launched independent experiments for each case, perfecting an interval around a point estimate until some convergence or time criterion was reached. Some experiments ran until meeting a confidence criterion, or were truncated upon exceeding an execution timeout; in these cases the measure of interest is the speed of

convergence. Notice this was the setting for all experiments in Section 3.5. Other experiments ran for a predefined execution time bound; in these cases the measure of interest is the precision achieved, where the goal is to build the narrowest possible interval.

Two computer systems were employed. The server *JupiterAce* features a 12-cores 2.40GHz Intel Xeon E5-2620v3 processor, with 128 GiB 2133MHz of available DDR4 RAM. The nodes of the cluster *Mendieta* count instead with 8-cores 2.70GHz Intel Xeon E5-2620 processors, each with access to a DDR3 RAM memory of 32 GiB and 1333MHz. For each case study we specify whether computations were done in JupiterAce or Mendieta. We point out however that FIG uses one core per estimation.

As indicated, some Markovian systems were imported from the previous chapter and analysed in the same way, that is, convergence was tested for decreasing values of the rare event probability γ . Remarkably this includes the database system with redundancy, which could not be evaluated thoroughly in Chapter 3 due to the limitations of the monolithic approach.

Two non-Markovian systems are also introduced in this section, whose analysis is carried out somewhat differently. The triple tandem queue that we present in Section 4.6.3 was tested for different configurations of its parameters, all of which yield roughly the same value of γ . Furthermore there are two variants of the oil pipeline system we study in Section 4.6.6: in one of them the system components fail according to exponentially distributed clocks; the other variant uses clocks which sample time from the Rayleigh distribution—i.e. a Weibull with shape parameter 2.

When applicable, we tested four simulation strategies for each model and configuration: standard Monte Carlo, RESTART using ad hoc importance functions, RESTART using the monolithic importance function from Chapter 3, and RESTART using the compositional approach from this chapter. Also, when the system model from the literature could be imitated exactly, we checked the consistency of the confidence intervals obtained by comparing them to the published values. Otherwise we verified that all simulation strategies converged to similar values, e.g. checking whether all intervals produced share a common region.

We present charts and tables displaying either the *convergence times* or the *precision of the intervals* obtained, depending on whether the execution bound was *confidence criteria* or *time budget* respectively. Time measurements consider wall-clock time, including preprocessings like the compilation of the model files and the selection of the thresholds.

In some cases the results evidence a high sensitivity to the choice of seed

fed to the Random Number Generator routine (RNG). This is exacerbated by the low replication we could perform: three to four independent experiments were run for each configuration of each case study, which presents a risk from the statistic viewpoint. Therefore, whenever unexpected or highly varying results were observed, extra replications were performed to verify consistency. This, on top of previous studies like the ones from [BDM17], accounts for the reliability of the measurements we present along this section.

It will be noted that generally, the simulation times obtained are longer than those presented before in Section 3.5, and also that the confidence convergence criteria is laxer. In that respect we highlight that a simulation step in PRISM involves accessing a matrix stored in memory, whereas in FIG all the clocks from all the modules have to be updated to perform the same task. This multiplication of floating point instructions per step is the price to pay for handling arbitrary distributions.

Moreover, FIG has so far been developed for correctness and not for efficiency. For instance, interval update is a trivially parallelisable task, yet FIG does it sequentially. This and several other tweaks could speed-up execution times significantly, but fall outside the goals of this thesis. In that sense the tool can be considered prototypical.

In any case, all convergence times presented along this thesis are used to compare the efficiency between the various strategies tested on each case study. To achieve this comparison it suffices to ensure that, for each case study, all the strategies being compared use the same hardware resources and terminate by the same convergence criterion. That was precisely the approach in Section 3.5 and also in this section. Obtaining faster executions with our software tools can be the subject of further research.

4.6.2 Tandem queue

We repeated the experiment previously presented in Section 3.5.2, using the IOSA model from Appendix A.6 to run simulations in JupiterAce. Recall this system consists in a Jackson tandem network with two sequentially connected queues, where the rates of arrival, first service and second service are respectively $(\lambda, \mu_1, \mu_2) = (3, 2, 6)$, and for which transient and steady-state properties were evaluated.

Given this is a Markovian system, the results yielded by FIG for the IOSA model from Appendix A.6 ought to coincide with those yielded by PRISM for an equivalent model written in its input language. One such model, effectively used to corroborate this claim, is presented in Appendix A.7.

Transient analysis

The property of interest is $P(q_2 > 0 \cup q_2 = c)$, i.e. the likelihood of observing a saturated second queue before it becomes empty, which we estimate starting from the state $(q_1, q_2) = (0, 1)$. We tested maximum queue capacities $C \in \{8, 10, 12, 14\}$, for which the values of γ approximated by PRISM[†] are respectively 5.62e-6, 3.14e-7, 1.86e-8, and 1.14e-9. Estimations were performed under a 90\40 CI criterion, that is, FIG had to build an interval with 90% confidence level and 20% relative error for each configuration. The execution timeout was 2.5 hours, within which FIG converged for each configuration producing intervals containing the values reported by PRISM.

Three different importance functions were tested in the importance splitting simulations. The function denoted `amono` was automatically built by FIG using the monolithic approach from the previous chapter. Instead, `acomp` stands for the function built following the compositional strategy, which in this case employed summation as composition operand. The third importance function tested with RESTART was the best ad hoc candidate from our previous studies, viz. counting the number of packets in the second queue, denoted `q2`. As before, standard Monte Carlo simulations are denoted `nospplit`.

The average of the wall times measured in three experiments are shown in Figure 4.4. Recall we display one chart per splitting value, with the outcomes of the `nospplit` simulations repeated in all four charts. The maximum queue capacity C , tuned to variate the rarity of the event, spans along the x-axis.

In accordance with our previous study of this system, standard Monte Carlo simulations could converge within the time limit only for the two smallest values of C , and they were always the slowest. Contrarily, RESTART simulations converged in all settings, with no clear winner among the three functions. In several configurations the times of function `q2` resemble that of `acomp`. Notice that the rare event property involves solely a variable from the second queue. Hence the local importance function of the first queue is *null*, turning `acomp` into something very similar to the ad hoc function.

The global tendency of the RESTART simulations favours the greatest splitting values. The technical output of FIG reveals that for each value of C , *the number of chosen thresholds does not increase as the splitting value decreases from 15 to 2*. This is undesirable, since spawning less offsprings upon crossing a threshold upwards should be countered with the placement of more thresholds. We believe this issue, from here onward denoted the

[†] Say the model from Appendix A.7 is in `tandem.prism`, then the exact command used is: `prism tandem.prism -const c=8:2:14 -pf 'P=?[q2>0 U q2=c]'`.

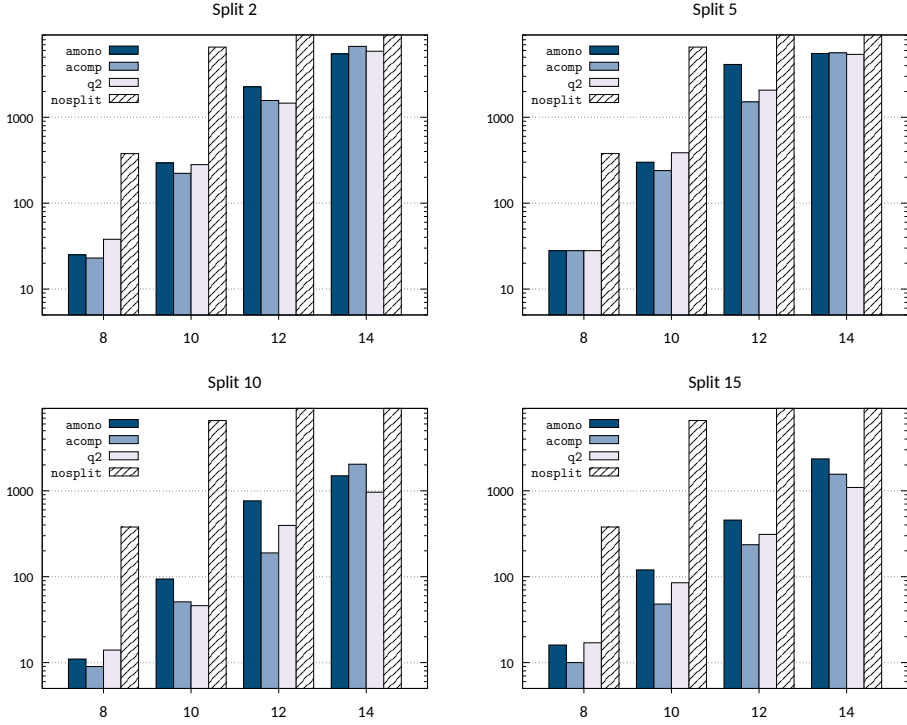


Figure 4.4: Times for the transient analysis of the tandem queue

splittings-thresholds fiasco, is related to the continuum assumptions of the theory behind Algorithm 5, and also to some implementation details of the algorithm in FIG. We will elaborate further on the subject along this section.

In spite of such issue, the plots show that the monolithic function was the least sensitive—though not by much—to the choice of global splitting value, whereas `acom` and `q2` converged the fastest for the splitting values 10 and 15.

The plots also reveal a relatively bad performance of `amono` w.r.t. `acom` and `q2`, most notably for the splittings 10 and 15 when $C = 10$, and also for splittings 5, 10, and 15 when $C = 12$. This is a most unwelcome surprise, since the monolithic approach outperformed all ad hoc functions in the results previously presented in Section 3.5.2. As discussed in that section, a superior performance of the monolithic approach is expected in these kind of systems: the queues are interconnected, hence all of them might need to be considered when deriving the importance function.

	$C = 10$		$C = 12$		
Splitting:	15	10	15	10	5
Num. Thr.:	3	5	4	6	8
amono	71 s	41 s	164 s	224 s	68 s
acomp	161 s	93 s	566 s	407 s	665 s
q2	137 s	86 s	657 s	395 s	594 s

Table 4.1: Convergence times for thresholds chosen ad hoc

New experiments were run for those configurations, to see whether this was related to the splittings-thresholds fiasco, and to discard the influence of the randomised seeds fed to the RNG. These experiments used a different, non-adaptive thresholds selection mechanism offered by FIG, tuned here to ensure that less splitting yields more thresholds. Table 4.1 details the outcomes: remarkably, the monolithic function outperformed the other two in *all* runs by factors ranging from 2x to 10x.

The results from Table 4.1 suggests that the monolithic function is actually the one performing best in this system, and that the slow convergence times from Figure 4.4 are mostly the result of the splittings-thresholds fiasco.

In any case this experiment shows that the compositional approach introduced in this chapter is fully functional, and that it can match other (very efficient) ad hoc approaches; all this without expanding the state space of the fully composed model, and without requiring the user to specify an importance function explicitly.

Steady-state analysis

Regarding long run simulations we are interested in the property $S(q2==c)$, i.e. the proportion of time that the second queue spends in a saturated state. We tested maximum queue capacities $C \in \{10, 13, 16, 18, 21\}$, for which the values of γ approximated by PRISM[‡] are respectively 7.25e-6, 2.86e-7, 1.12e-8, 1.28e-9, and 4.94e-11. Estimations with FIG had to converge within 6 hours of wall time execution achieving a 90\40 CI. Again we corroborated that these estimations converged to the values yielded by PRISM. The same importance functions as in the transient case were employed.

The results obtained from an average among three experiments are pre-

[‡] `prism tandem.prism -const c=10:3:21 -pf 'S=?[q2=c]' -jor.`

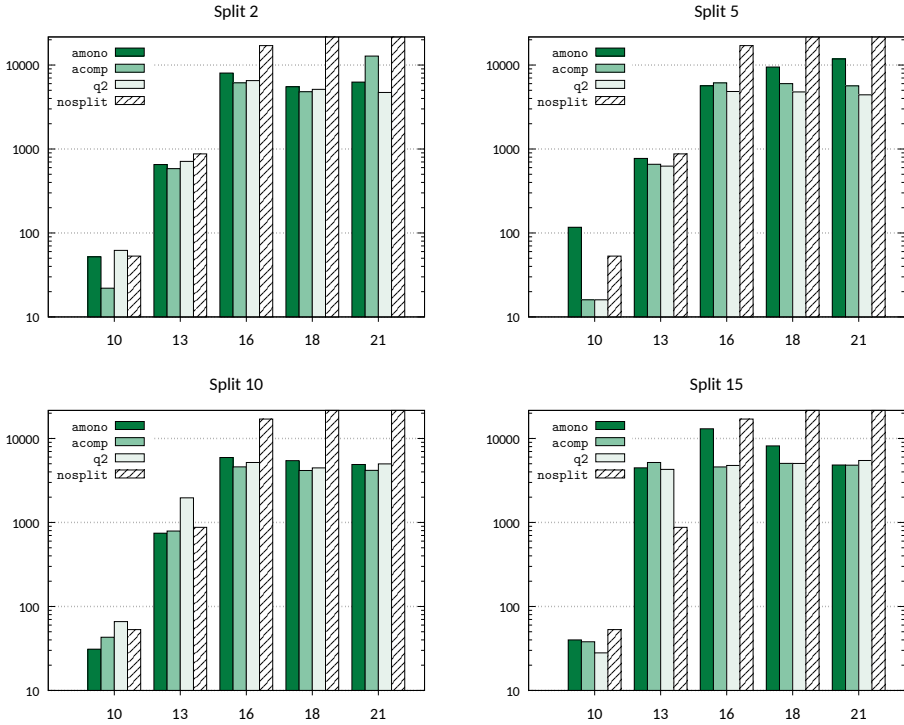


Figure 4.5: Times for the steady-state analysis of the tandem queue

sented in Figure 4.5, following the same format than in the transient case. The queue capacities tested (as well as the time limit) differ from the ones used in Section 3.5.2, yet the behaviour of the standard Monte Carlo simulations is quite similar, converging reasonably fast only when $C < 15$. In contrast, none of the RESTART simulations failed to meet the confidence criterion within the time limit.

Leaving aside the case of $C = 10$, which anyway is the least rare and thus the least interesting, convergence times are rather uniform for all importance functions and for splittings 2, 5, and 10. We would have expected a better performance of the monolithic approach w.r.t. `acom` and `q2`, but as before we believe that the splittings-thresholds fiasco is creating a distortion. In particular this caused the anomaly of `amono` for splitting 15 and $C = 16$. The specific problem behind such behaviour is detailed in Section 4.6.4.

Unfortunately, this distorting issue cannot be countered systematically without a deep refactoring in the thresholds selection mechanisms of the FIG

tool. We refer to this matter again in the concluding remarks of the thesis.

Be that as it may, the compositional approach performed very well, once again suggesting that our technique can automate the derivation of a high quality importance function, without expanding the state space of the fully composed model.

As last remark we note that the shape of the (importance splitting) plots resembles a logarithmic grow in the convergence times. Since the y-axis is in log-scale this would suggest that RESTART is showing logarithmic efficiency, with convergence times growing sub-exponentially as the rarity of the event grows exponentially. This was not the case in the previous transient study, where convergence times appear to grow exponentially, inversely to the exponential decay of γ .

In that respect we note that RESTART was primarily devised for steady-state studies—see [VAVA91, VA98, VAVA02, VA14]. In transient cases where simulations need to be respawned at high rates, the costs incurred in the global splitting mechanisms of RESTART may not pay off. Instead, strategies like Adaptive Multilevel Splitting could lead to better results, where simulation paths are spawned and truncated in a stepwise approximation to the set of rare states. In Section 5.1 we briefly revisit the subject of implementing other splitting simulation mechanisms in FIG.

4.6.3 Triple tandem queue

Consider a non-Markovian tandem network operating under the same principles than the tandem queue from the previous section, but consisting of *three* queues with *Erlang-distributed* service times[§]. The shape parameter α is the same for all servers, but the scale parameters $\{\mu_i\}_{i=1}^3$ differ from one queue to the next. Arrivals into the system are exponential with rate λ .

The long run behaviour of this non-Markovian triple tandem queue was studied in [VA09] starting from an empty system. The shape parameter is $\alpha \in \{2, 3\}$ in all queues and the load at the third queue is kept at $1/3$. This means that the scale parameter μ_3 in the third queue takes the values $1/6$ and $1/9$ when α is 2 and 3 respectively. The scale parameters μ_1 and μ_2 of the first and second servers, as well as the thresholds capacity C at the third queue, are chosen to keep the steady-state probability in the same order of magnitude for all case studies.

[§] Although this could be emulated using the exponential distribution, we will maintain a non-Markovian approach.

We use the IOSA model presented in Appendix A.8 to run simulations in JupiterAce. The property of interest is the steady-state probability of a saturation in the third queue, i.e. $S(q3==c)$. Following the same approach from [VA09] we choose a value of γ in the order of $5 \cdot 10^{-9}$. Thus the values of $(\alpha, \mu_1, \mu_2, C)$ for the six case studies I–VI are

$$\begin{array}{ll} \text{I: } (2, 1/3, 1/4, 10) & \text{IV: } (3, 1/9, 1/6, 9) \\ \text{II: } (3, 2/3, 1/6, 7) & \text{V: } (2, 1/10, 1/8, 14) \\ \text{III: } (2, 1/6, 1/4, 11) & \text{VI: } (3, 1/15, 1/12, 12). \end{array}$$

Estimations had to achieve a 90\40 CI within 4 hours of execution. Four importance functions were tested in the splitting simulations: the monolithic (`amono`) and compositional (`acomp`) functions which FIG can build automatically, using summation as composition operand for `acomp`; an ad hoc function which just counted occupation in the third queue (`q3`); and the ad hoc approach (`jva`) from [VA09], which also considers the occupancy in the other queues with weight coefficients—in the interval $[0.2, 0.9]$ —specific to each case.

Results are presented in Figure 4.6. This experiment was also run three times; the values in the plots show the average of the convergence times measured. Case studies I–VI span along the x-axis of each plot.

As expected, standard Monte Carlo simulations failed to converge within the 4 hours limit imposed in almost all cases. On the other hand RESTART simulations converged in time in most settings, yielding an interval estimate with the desired properties.

Case study II is quite curious because it was the only one in which `nosplit` simulations converged, estimating the interval $[6.13e-8, 9.20e-8]$. Besides, not only did they converge, but did so (with few exceptions, perhaps most notably `amono`) *faster than many RESTART simulations*. This is not fortuitous and can be explained as follows:

- II is the least rare case study; compare it to III where point estimates are around $1.69e-8$, or V where they are around $3.40e-9$, i.e. roughly one order of magnitude smaller.
- II is the setting with the smallest queue capacity (e.g. V uses a value of C twice as big as II), and thus the setting where splitting can produce the least gain.

These arguments notwithstanding, there was always at least one RESTART simulation outperforming standard Monte Carlo. See for instance `amono` for

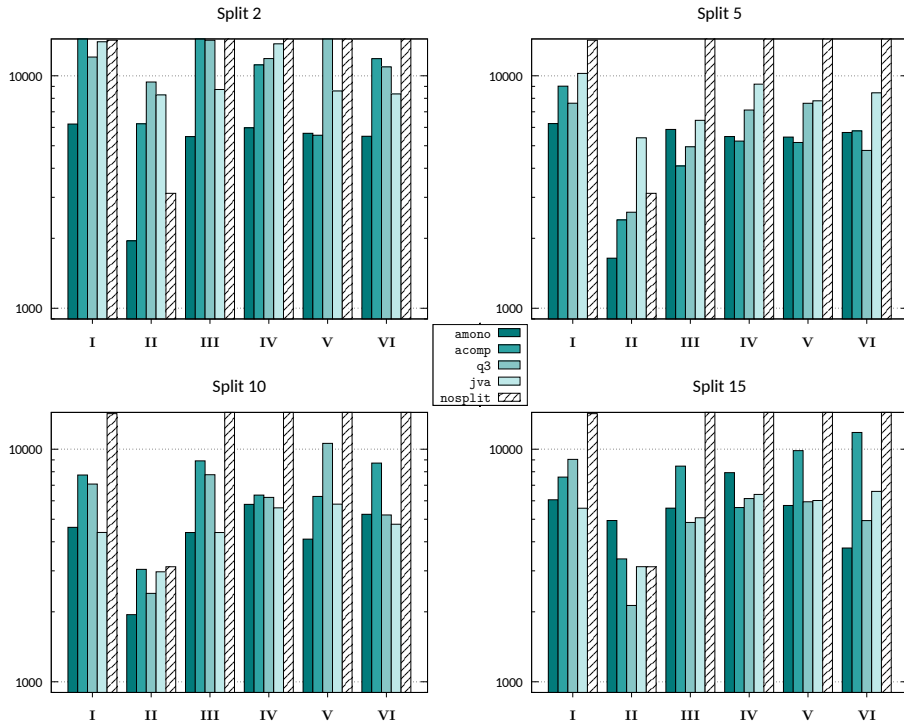


Figure 4.6: Times for the steady-state analysis of the triple tandem queue

splittings 2, 5, and 10, and also q3 for splittings 5, 10, and 15. In particular for splitting 10 all RESTART simulations converged faster than `nosplit`.

Unfortunately, just like it happened with the tandem queue system, there is much variability in the results for the different global splitting values tested. This is most notable for splitting 2, where RESTART simulations converged the slowest, some of them even producing timeouts in all three experiments ran—see cases **I**, **III**, and **V**. Most importance functions converged the fastest for the splitting value 10.

In spite of these problems with the global splitting, which are intimately related to the selection of the thresholds as explained, these experiments with the triple tandem queue contributed in three major ways:

1. even though the system can be encoded to fit in a Markovian setting, our model employs the Erlang distribution, allowing a more compact representation and also proving our claims regarding the generality of our techniques and algorithms;

2. once again the compositional approach performed quite well, showing that a full state space expansion can be unnecessary to automate the construction of a reasonable importance function;
3. `amono` always finished on time, and in almost all scenarios was either the fastest or the runner up, which coincides with our expectations as previously detailed in Section 4.6.2.

4.6.4 Queueing system with breakdowns

We repeated the experiment presented in Section 3.5.5 and originally studied in [KN99], consisting in a queueing network where several sources (of types 1 and 2) send packets to a single buffer attended by a server. All sources, as well as the server, can break down and get repaired later on. The system is Markovian with rates of component repair, component failure, and packet production/processing $(\alpha_1, \beta_1, \lambda_1) = (3, 2, 3)$, $(\alpha_2, \beta_2, \lambda_2) = (1, 4, 6)$, and $(\delta, \xi, \mu) = (3, 4, 100)$ respectively for sources of type 1 and 2 and for the server.

We use the IOSA model from Appendix A.9 to study the transient behaviour of the system by running experiments in JupiterAce. More precisely we are interested in the probability of observing a saturated buffer before it becomes empty, starting with a single buffered packet and all system components broken but for one source of type 2. The corresponding property query is `P (!reset U buf==K)`.

We studied this system for the buffer capacities $K \in \{40, 60, 80, 100\}$, with corresponding values of γ equal to 4.59e-4, 1.25e-5, 3.72e-7, 9.59e-9. These values were approximated by PRISM in the equivalent CTMC from Section 3.5.5 (see Appendix A.4), and the convergence of FIG to such values was checked for all settings. In particular estimations had to achieve a 90\40 CI within 3 hours of wall time execution. Three importance functions were tested in the splitting simulations, namely the monolithic (`amono`) and compositional with summation (`acomp`) functions built by FIG, and the best ad hoc variant resulting from our studies in Section 3.5.5, i.e. counting the number of packets in the buffer (`buf`).

Figure 4.7 shows the average times to convergence, obtained from four experiments run with FIG. The behaviour of the standard Monte Carlo simulations resembles the previous experiments with BLUEMOON, where it had converged only for $K < 80$. This time however, for $K = 40$, it took `nosplit` simulations about three minutes to build an interval with the desired properties. With the CTMC model of the queue with breakdowns running on

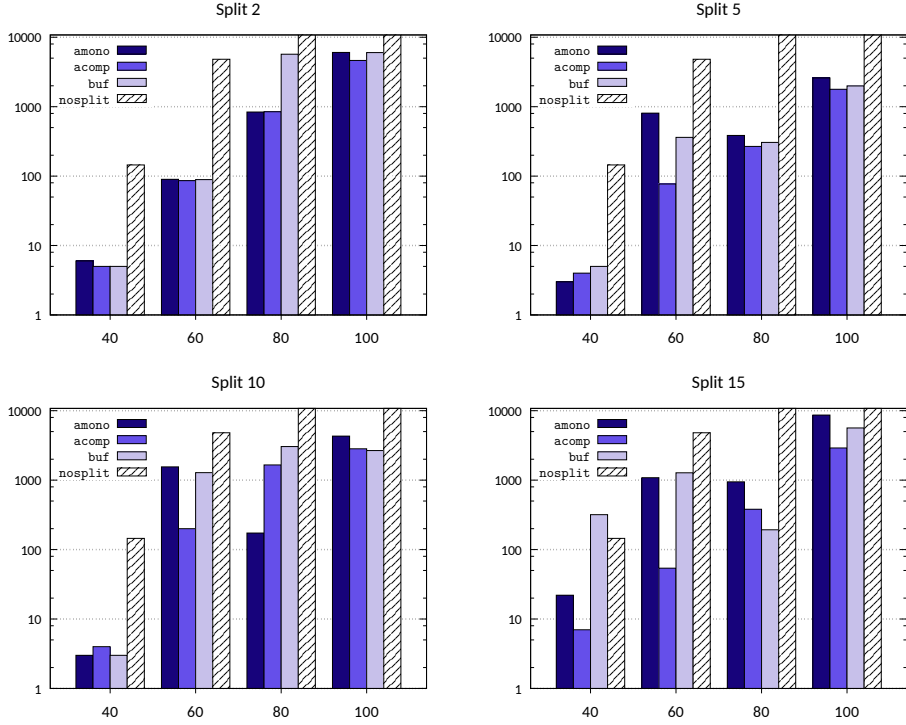


Figure 4.7: Times for the transient analysis of the queue with breakdowns

BLUEMOON it had taken less than 10 seconds. Clearly and as expected, updating several clocks in several modules (e.g. like in IOSA) has worse performance than accessing a matrix (e.g. like in a CTMC transitions representation).

Moreover this provided an advantage for the splitting simulations, since the sheer brute force of `nosplit` is then less advantageous than selecting (proper) resplitting points like `RESTART` does. As a consequence, for $K = 40$, all but one of the splitting configurations (viz. `buf` for splitting 15) were faster to converge than standard Monte Carlo.

Once again the high variability of the results for the different global splitting values complicates a clean comparison among the three importance functions. Overall however none of them clearly outperformed the rest. We use this, as we have done before, to highlight that the compositional approach can yield reasonable results even in settings where a monolithic function has a theoretic advantage.

A striking peculiarity of Figure 4.7 are the incongruously long convergence

times of: `buf` for $K = 40$ and splitting 15 (one configuration); and both `buf` and `amono` for $K = 60$ and splittings 5, 10, and 15 (six configurations). In particular for $K = 60$, five out of these six configurations took *longer* to converge than for $K = 80$.

Studying the technical output of FIG reveals that from the thresholds selected in those cases, half or more of them were actually not chosen by the adaptive component of Algorithm 5. At some point in those experiments, an iteration of Sequential Monte Carlo had failed to find a higher threshold, and the algorithm fell back to the deterministic selection provided by `choose_remaining(...)`. This was observed in 1–3 out of the 4 experiments run for those configurations. Precisely those experiments were the ones yielding the incongruously long convergence times reported.

Evidently, such behaviour is problematic. The best efforts are made to mimic an intelligent selection of thresholds in `choose_remaining(...)`, taking into account the splitting value used, the post-processing (if any), and the importance range left to cover after (our implementation of) Sequential Monte Carlo has failed. However, `choose_remaining(...)` implements a deterministic selection, which does not consider the stochastic behaviour of the model like only an adaptive algorithm can.

To observe this *early fail* of the adaptive component in Algorithm 5, which we identify as the main cause behind the splittings-thresholds fiasco, leads us to believe that a different approach should be sought. Some reflections on a potential solution are outlined in Section 5.1.

4.6.5 Database system with redundancy

Recall the model of a database facility introduced in Example 7 to show the limitations of the monolithic approach. The system has a characterising redundancy $R \in \mathbb{N}$ ($R > 1$) and its components are processors (two types of them), disk controllers (two types of them), and disk clusters (six of them). Denoting by *unit* any type of processor, any type of controller, and any disk cluster (i.e. there are ten units in total), the system is operational inasmuch less than R components have failed in any unit.

This Markovian database was originally studied in [GSH⁺92] and then using RESTART in e.g. [VA98]. The failure rates of processors, controllers, and disks are respectively μ_P , μ_C , and μ_D . Furthermore all these components can fail with equal probability in one of two types: failures of type 1 involve a repair rate equal to 1.0, and those of type 2 have a repair rate of 0.5. Rare event analyses focus on system unavailability, e.g. γ reflects the proportion

of time the database is not operational.

We ran experiments in Mendieta with models like the one presented in Appendix A.10, which describes (a summarised version of) a system for redundancy $R = 2$. The property at the bottom queries the steady-state probability of having any two (R) components failed in the same unit:

$$S \left((d11f \ \& \ d12f) \mid (d11f \ \& \ d13f) \mid \dots \mid (p21f \ \& \ p22f) \right) .$$

Since the IOSA formalism associates a single probability density function with each clock, the inter-processor failure hypothesis is dropped (cf. [VA98]). Moreover, our IOSA models have repair clocks individual to each component, in contrast to the single (sequential) repairman scheme from [VA98, VA07a].

We studied systems with failure rates $(\mu_P, \mu_C, \mu_D) = (1/50, 1/50, 1/150)$ and redundancy values $R \in \{2, 3, 4, 5\}$. Due to the long times FIG took to converge for the larger models, this experiment follows a different scheme than those presented before. Rather than requesting estimations to achieve a predefined confidence criterion, we impose an execution time budget, and measure the precision of the intervals built by each strategy at timeout. The goal is to estimate the narrowest possible interval in the available time, where the time budgets for the redundancy values $R = 2, 3, 4, 5$ are 10 seconds, 2 minutes, 20 minutes, and 6 hours respectively.

We highlight that our models are incomparable to those studied by [GSH⁺92, VA98, VA07a] due to the different hypotheses we use in order to model the database with IOSA. Besides, even though the database is Markovian, PRISM cannot be utilised to approximate the results in an equivalent CTMC, owing to the physical memory issues reported in Section 3.6. As a workaround, to verify correctness in our estimations we compared the confidence intervals yielded by all runs, corroborating that they share a common region. The mean values (and the standard deviation) thus obtained from the central point estimates for each redundancy are:

$R :$	2	3	4	5
avg $\{\hat{\gamma}_i\}$:	6.86e-3	5.14e-5	3.81e-7	8.06e-9
stdev $\{\hat{\gamma}_i\}$:	1.46e-4	2.34e-5	4.37e-7	1.49e-8.

Five importance functions were tested with RESTART. The monolithic approach is ruled out due to the memory issues addressed. The compositional approach with summation as composition operand is denoted `ac1`. Since each component can be either failed or operational, its local importance function will take values 1 and 0 respectively. Hence `ac1` counts the number of failed

components in the system. In spite of its simplicity, this strategy builds a much richer importance structure than the monolithic approach could.

The compositional function denoted **ac2** makes a distinction based on the component type, under the hypothesis that their failure rates may be different and thus they should not be mixed up (cf. **ac1**). Using the exponentiation post-processing, the local importance functions of all disks are multiplied together ($F_D \doteq \prod_{i,j=1,1}^{6,R+2} D_{i,j}$), and the same is done with all controllers ($F_C \doteq \prod_{k,\ell=1,1}^{2,R} C_{k,\ell}$) and all processors ($F_P \doteq \prod_{k,\ell=1,1}^{2,R} P_{k,\ell}$). The global importance function is the sum of these values: $F_D + F_C + F_P$.

Function **ac3** makes an even finer distinction, separating the product of the disks per cluster ($F_D^i \doteq \prod_{j=1}^{R+2} D_{i,j}$ for clusters $1 \leq i \leq 6$), and of the controllers and of the processors per type ($F_C^k \doteq \prod_{\ell=1}^R C_{k,\ell}$ and $F_P^k \doteq \prod_{\ell=1}^R P_{k,\ell}$ for types $1 \leq k \leq 2$). Again, the global importance function is the sum of these values: $\sum_{i=1}^6 F_D^i + \sum_{k=1}^2 F_C^k + \sum_{k=1}^2 F_P^k$.

Function **ac4** uses the $(+, *)$ ring in what can be regarded as a further refinement in the same direction than **ac2** and **ac3**: **ac2** would be the most coarse grained, mashing all components of the same type together; **ac3** contemplates the divisions of the system in independent units; and **ac4** distinguishes every possible configuration leading to a system failure.

Finally, function **ah** has two faces. On the one hand it can be regarded as a compositional variant implementing the $(\max, +)$ semiring. On the other hand it matches exactly the ad hoc proposal of [VA07a], where the function is denoted $\Phi(t) \doteq cl - oc(t)$.

For a confidence level of 90%, the precision of the intervals obtained for the time budgets reported are presented in Figure 4.8. These values are the average of the outcomes from four independent experiments run in Mendieta.

Unsurprisingly, standard Monte Carlo simulations yielded the best interval estimates for the lowest redundancy, $R = 2$, coinciding partially with the results reported in [BDM17]. Notice however that for $R = 3$ and although to a moderate extent, Figure 4.8 shows a better behaviour of all RESTART simulations w.r.t. **nospplit**, unlike in [BDM17].

Even though these outcomes can be belittled as yet another example where the event is not rare enough for a really effective application of multilevel splitting (which might well be true!), some insight can be gained from a deeper analysis of the situation.

To become non operational the database requires R components in any unit to fail, where the number of components per unit is heterogeneous. The unit with most components would be the most likely to produce the system

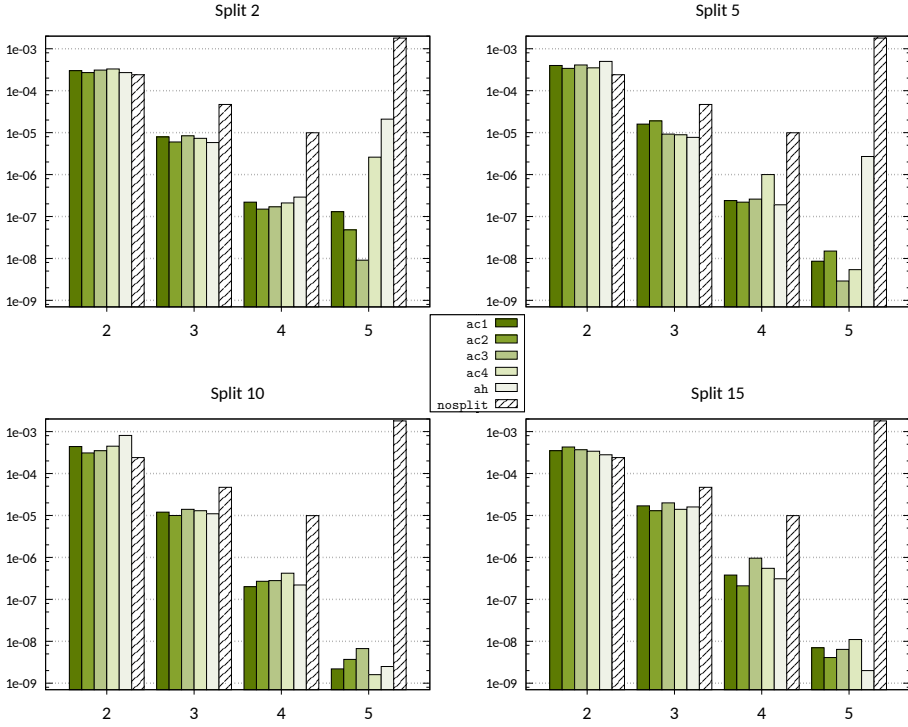


Figure 4.8: Intervals precision for the steady-state analysis of the database

failure, which is the case of the disks clusters. Nevertheless, the lifetime among components differs greatly, and on average it is three times shorter in controllers and processors than in disks. This means that it is actually very likely to observe a non operational database due to R failed controllers or processors from the same unit, i.e. of the same type.

That is why one *needs* to increase the redundancy value in order to obtain some gain from the use of multilevel splitting. Even in the rich layering offered by ac1 to ac4, most cases of a non operational database will be caused by R failed processors or controllers of the same type. Which does not imply, however, that the compositional approach is at a loss. It merely justifies why nosplit performed better than the splitting simulations for $R = 2$. Higher redundancies mean a more layered structure of even the most likely, flattest failures, which goes in favour of using multilevel splitting.

We now draw attention to redundancy values $R \in \{4, 5\}$, where standard Monte Carlo ceases to be a reasonable choice and one has to resort to

other strategies, e.g. importance splitting. Moreover, since the monolithic approach was infeasible already for $R = 4$ in our studies from Section 3.6, the compositional approach introduced in this chapter offers the only automatic alternative to apply multilevel splitting.

Remarkably, none of the five composition strategies clearly outperformed the rest in all configurations. This would suggest that, in this scenario where *flat failures* caused by R processors or controllers have great likelihood, the extra importance layering granted by functions like `ac4` is not a defining factor. The charts also hint at a better performance of `RESTART` for the higher splitting values, although convergence was slightly faster in almost all cases for splitting 10 than for splitting 15.

A few configurations yielded values amiss the rest; the most striking cases from Figure 4.8 are observed when $R = 5$ for splitting 2 (`ac4` and `ah`) and splitting 5 (`ah`). Furthermore, for that redundancy and in one out of the four independent experiments run, `ac4` and `ac1` failed to converge to an estimate for splittings 10 and 15 respectively. These peculiarities are, to our surprise, *not related* to the splittings-thresholds fiasco. The technical output of FIG revealed an aberration in the outcome of the individual simulations, which at some point started to sample the rare event at extremely high (or low) rates, contrarily to the immediately preceding behaviour. Suspecting a relation between these aberrations and the pseudo-random number generation algorithm, we repeated such runs using a different algorithm fed with different seeds. As expected, the aberrations were not observed again, and the outcomes fitted the normal setting corresponding to each case.

As last remark we highlight that in spite of the similar performance among all importance functions, both `ac4` and `ah` (the last regarded as the $(\max, +)$ semiring) are not specifically designed for the database, but can be derived from the DNF expression of the rare event as described in Section 4.3.3. That is one good reason to prefer them above the other three.

4.6.6 Oil pipeline

Consider a consecutive- k -out-of- n : F system, usually denoted $C(k, n: F)$. This consists of a sequence of n components *ordered sequentially*, so that the whole system fails if k or more *consecutive* components are in a failed state. For a more down-to-earth mental picture consider an oil pipeline where there are n equally spaced pressure pumps. Each pump can transport oil as far as the distance of k pumps and no further. Thus if $k > 1$ then the system has certain resilience to failure, and remains operational as long as no k

consecutive pumps have failed, otherwise regardless of how many pumps have failed.

$C(k, n: F)$ systems have been studied as early as 1980 [Kon80]. Several generalisations exist to the original setting; we are interested in the non-Markovian and repairable systems analysed in e.g. [XLL07, VA10]. Those works assume the existence of a repairman which can take one failed component at a time and leave it “as good as new,” after a log-normal distributed repair time has elapsed [XLL07]. In particular [VA10] consider also the existence of non-Markovian failure times (namely, sampled from the Rayleigh—or Weibull—distribution) and measure the steady-state unavailability of the system. Notice the probability density function used in [VA10] for the Rayleigh distribution is $f_\beta(t) \doteq \beta t e^{-t^2 \frac{\beta}{2}}$, whose mean is $\sqrt{\frac{\pi}{2\beta}}$.

To run the experiments in Mendieta we use IOSA models like the one shown in Appendix A.11, which represents an oil pipeline of the type $C(3, 20: F)$, i.e. where there is a total of $n = 20$ pressure pumps, and $k = 3$ consecutive failed pumps cause a general system failure. In that example the steady-state system unavailability is given by the property query:

$$\begin{aligned} & S ((\text{broken1} > 0 \ \& \ \text{broken2} > 0 \ \& \ \text{broken3} > 0) \mid \\ & \quad (\text{broken2} > 0 \ \& \ \text{broken3} > 0 \ \& \ \text{broken4} > 0) \mid \\ & \quad \quad \quad \vdots \\ & \quad (\text{broken18} > 0 \ \& \ \text{broken19} > 0 \ \& \ \text{broken20} > 0)) . \end{aligned}$$

Unfortunately the $(k - 1)$ -step Markov dependency of the sources (see [LZ00] and also [XLL07, VA10]) cannot be modelled in IOSA, since it would require to associate more than one distribution to the clocks involved[†]. We also highlight that in order to model the repairman, an extension of the basic IOSA theory and model syntax presented in Sections 4.4 and 4.5.2 was employed, which allows certain use of instantaneous (or untimed) actions[§]. That, plus the possibility to define and operate with arrays of variables, is currently supported by the FIG tool version 1.1.

Still, there is no support in FIG for the repair policies reported in [VA10]—we point out that the tool is designed to fit a general basis of models, and such repair policy is quite singular to this system. This issue, plus the absence of the $(k - 1)$ -step Markov dependence hypothesis, make our implementation

[†] Several extensions to the formalism are under consideration; this is one of them.

[§] This is ongoing research by Monti et al. at the Dependable Systems Group.

of the models incomparable to those studied in [XLL07, VA10]. Therefore we resort to the same strategy followed with the database, and for each system configuration studied we report the mean value of all probabilities estimated, as well as their standard deviation.

Specifically, we studied models with $n \in \{20, 40, 60\}$ sequentially ordered components, where $k \in \{3, 4, 5\}$ sequential failures result in a non operational system. As in [VA10] we analysed both exponential and Rayleigh failure times, for rate and scale parameters $\lambda = 0.001$ and $\beta = 0.00000157$ respectively. This yields the same mean lifetime in the components. Repair time is sampled from a log-normal distribution with parameters $\mu = 1.21$ and $\sigma = 0.8$. The steady-state system unavailability estimated for these configurations is shown in Table 4.2.

		Exponential			Rayleigh		
		20	40	60	20	40	60
$k=3$	avg $\{\hat{\gamma}_i\}$:	1.53e-5	4.65e-5	1.10e-4	2.02e-5	6.54e-5	1.63e-4
	stdev $\{\hat{\gamma}_i\}$:	1.76e-6	4.68e-6	1.62e-5	2.49e-6	7.27e-6	2.21e-5
$k=4$	avg $\{\hat{\gamma}_i\}$:	2.92e-7	1.49e-6	4.33e-6	5.44e-7	2.65e-6	8.16e-6
	stdev $\{\hat{\gamma}_i\}$:	1.39e-7	2.51e-7	6.64e-7	1.57e-7	4.04e-7	1.06e-6
$k=5$	avg $\{\hat{\gamma}_i\}$:	2.62e-9	5.93e-8	3.06e-7	7.49e-9	1.26e-7	6.97e-7
	stdev $\{\hat{\gamma}_i\}$:	2.03e-9	2.54e-8	1.39e-7	6.11e-9	4.49e-8	2.77e-7

Table 4.2: Unavailability estimates for the oil pipeline

We performed two independent experiments, each covering all eighteen configurations, running FIG in Mendieta and requesting a 90\40 CI. We imposed differentiated wall time execution limits for the different values of k , since this parameter has the highest influence in the rarity of the event (see Table 4.2) and thus in the convergence times. For $k = 3, 4, 5$ we requested estimations to converge within 1.5, 3, and 6 hours respectively.

The situation with the oil pipeline models is similar to the database, in the sense that the high amount of components (which fail independently from each other) render any monolithic approach utterly infeasible. Therefore the automatic importance functions tested with RESTART are compositional.

The naïve strategy of composing the local functions with summation

as composition operand is denoted `ac1`. Similarly, `ac2` uses product as composition operand and an exponentiation post-processing. The $(\max, +)$ and $(+, *)$ semiring and ring composition strategies are employed by the functions denoted `ac3` and `ac4` respectively. Last, `ah` uses the ad hoc interface of FIG to implement the $(\max, +)$ semiring, using the variables of the modules (i.e. in an ad hoc fashion) rather than the local importance functions which the tool could compute if requested. This is the approach followed in [VA10] and denoted $\Phi(t) \doteq cl - oc(t)$ in that work.

This case study features eighteen different configurations, each of them tested with standard Monte Carlo and with RESTART using five different importance functions. In addition, each multilevel splitting run was tested for the usual four different global splitting values. The average times to convergence in seconds are presented in Tables 4.4 and 4.5.

Sometimes only one out of the two experiments run for each setting produced a result; such single-simulation results appear enclosed in parentheses in Tables 4.4 and 4.5. Since no simulation converged within the six hours time bound for $K = 5$, in that cases we show the interval precision achieved at timeout for a 90% confidence interval. Also and as before we divide the charts per splitting value.

Since there are so many different configurations (eighteen in total), and also so many different settings tested for each configuration (a standard Monte Carlo run plus twenty RESTART runs if we tell apart by splitting and importance function), some simplifications are due to interpret these results. Table 4.3 present a very coarse filter, where we count the total number of RESTART simulations outperforming the standard Monte Carlo runs on each configuration.

	Exponential			Rayleigh		
	20	40	60	20	40	60
$k = 3$	9	8	0	3	3	0
$k = 4$	19	16	4	20	16	2
$k = 5$	7	15	6	10	10	4

Table 4.3: RESTART vs. Monte Carlo

We highlight that in Table 4.3, any splitting setting for which a single experiment (out of the two) converged, was considered to have lost against

		Split 2					Split 5				
n	k	ac1	ac2	ac3	ac4	ah	ac1	ac2	ac3	ac4	ah
	3	58	51	99	62	804	75	52	53	50	54
20	4	2780	2109	4398	2339	7257	6292	3535	4134	2683	3026
	5	(7.0e-10)	-	2.4e-9	(3.9e-9)	(1.7e-9)	5.6e-9	4.4e-9	(1.6e-9)	2.2e-9	2.9e-9
	3	105	1386	1388	153	132	111	1833	123	107	200
40	4	3439	3480	3590	10800	3902	4047	6344	5120	10800	3368
	5	5.8e-8	5.6e-8	4.5e-8	5.6e-8	3.8e-8	7.4e-8	5.1e-8	4.9e-8	2.1e-8	4.7e-8
	3	148	156	240	1953	256	274	383	404	202	438
60	4	10800	10800	4879	4023	4932	10800	10800	10246	4104	10800
	5	2.1e-7	1.8e-7	2.8e-7	1.7e-7	2.4e-7	3.8e-7	5.1e-7	5.3e-7	3.6e-7	3.2e-7

		Split 10					Split 15					nosplit
n	k	ac1	ac2	ac3	ac4	ah	ac1	ac2	ac3	ac4	ah	
	3	77	53	77	44	81	88	131	74	73	80	63
20	4	10800	4019	5095	3347	4922	5084	8445	6205	4336	3452	10800
	5	3.7e-9	2.1e-9	5.8e-9	2.6e-9	3.3e-9	(5.1e-10)	3.4e-9	7.0e-9	(1.7e-9)	5.1e-9	3.7e-9
	3	89	1456	221	66	137	92	1811	147	121	155	142
40	4	3868	3568	4072	10800	5973	4449	4332	4182	10800	5152	10800
	5	5.9e-8	4.4e-8	5.3e-8	5.3e-8	5.9e-8	3.9e-8	7.1e-8	1.3e-7	8.4e-8	9.5e-8	6.2e-8
	3	263	240	649	225	765	286	245	214	240	180	137
60	4	10800	10800	2799	5430	3502	10800	10800	5277	6135	5838	4332
	5	2.5e-7	1.5e-7	1.2e-7	2.3e-7	3.6e-7	2.8e-7	1.7e-7	2.4e-7	5.0e-7	2.9e-7	2.3e-7

Table 4.4: Results for the oil pipeline with exponential failures

n	k	Split 2					Split 5					
		ac1	ac2	ac3	ac4	ah	ac1	ac2	ac3	ac4	ah	
20	3	50	60	57	63	644	34	51	55	48	52	
	4	2884	2959	6625	2342	3775	4513	1673	2634	1598	1785	
	5	(1.5e-9)	5.0e-9	-	(4.0e-9)	(2.4e-9)	4.0e-9	(5.3e-9)	5.6e-9	(4.0e-9)	6.9e-9	
40	3	71	1504	139	123	193	119	1579	120	141	101	
	4	5140	3881	3326	10800	2000	2539	3720	2973	10800	5467	
	5	9.8e-8	1.4e-7	1.1e-7	9.3e-8	7.6e-8	8.6e-8	1.2e-7	1.4e-7	9.4e-8	8.2e-8	
60	3	256	222	185	2296	163	316	283	319	345	368	
	4	10800	10800	3218	3801	4528	10800	10800	7697	6488	5634	
	5	6.2e-7	3.2e-7	4.4e-7	2.9e-7	4.5e-7	4.6e-7	3.0e-7	4.5e-7	1.4e-6	4.8e-7	

n	k	Split 10					Split 15					nosplit
		ac1	ac2	ac3	ac4	ah	ac1	ac2	ac3	ac4	ah	
20	3	51	53	61	53	61	76	83	74	46	76	49
	4	2322	2172	1687	1645	3420	2584	3418	2740	3241	3008	10800
	5	-	1.3e-8	1.2e-8	(4.2e-9)	7.4e-9	(2.6e-9)	2.1e-8	1.7e-8	(4.5e-9)	7.8e-9	(2.3e-9)
40	3	92	1588	119	83	161	141	1893	123	106	111	101
	4	3143	3384	3558	10800	2613	3674	4510	3911	10800	4424	10800
	5	9.6e-8	1.0e-7	6.9e-8	9.1e-8	7.6e-8	1.5e-7	1.3e-7	1.4e-7	6.3e-8	9.4e-8	9.5e-8
60	3	433	239	450	254	609	387	528	1192	430	964	100
	4	10800	10800	10800	6726	9515	10800	10800	10800	7837	10800	3769
	5	9.7e-7	5.2e-7	3.2e-7	5.5e-7	1.8e-6	1.2e-6	7.7e-7	5.7e-7	8.7e-7	1.0e-6	3.7e-7

Table 4.5: Results for the oil pipeline with Rayleigh failures

standard Monte Carlo. This might be regarded as biased against multilevel splitting, but the only properly unbiased way of comparing all simulation settings would be to perform several more repetitions of the full experiment. Unfortunately, the long execution time of a full experiment[†] leaves this out of the question.

The above notwithstanding, in the case of systems with Rayleigh failure times we observe that using multilevel splitting pays off when the probability of the rare event lies below approximately $5.0\text{e-}6$, i.e. for $n \in \{20, 40\}$ when $k = 4$, and for all n when $k = 5$ (the cases when $n = 60$ are addressed next in more detail). Something quite similar happened with the exponentially distributed failure times, but for a lower magnitude, namely around $2.0\text{e-}6$.

It is also noteworthy that the general trend of Table 4.3 indicates higher values of n are detrimental to multilevel splitting w.r.t. standard Monte Carlo. This could be due to the fact that, in our models, the higher the value of n the lower the rarity of the event.

Even though that could indeed explain a *smooth* variant of such overall behaviour regarding n , there seems to be a *harder barrier* between the values 40 and 60 of n than between the values 20 and 40. In that sense notice that the outcomes in Tables 4.4 and 4.5 show several RESTART runs reached the maximum corresponding time bound—*timed-out*—thus failing to outperform the competing `nospplit` runs. The higher the value of n the more frequent this is observed; see e.g. the cases when $n = 60$ and $k = 4$.

If we take into consideration the splittings-thresholds fiasco, then we find a plausible explanation for this behaviour, where systems with $n = 60$ components are hard to analyse for FIG using RESTART. Namely, a higher n implies longer simulation steps since more clocks need to be updated per step. If the thresholds for multilevel splitting are selected poorly, the wasted time increases with the splitting. This should be exacerbated by having higher splitting values, although not necessarily in a linear way, since the quality of the selected thresholds plays a major role.

In the spirit of the above, for the configuration $n = 60$ and $k = 4$, for splitting values 2, 5, 10, and 15 respectively, Tables 4.4 and 4.5 show that out of the five RESTART settings: respectively 2, 2, 3, and 4 settings timed-out for Rayleigh distributed failure times; and respectively 2, 3, 2, and 2 settings timed-out for exponentially distributed failure times.

In any case we observe from Table 4.3 that the only configurations where, for any splitting, none or very few RESTART runs outperformed the

[†] It takes 6 days to test the 18 configurations with all simulation settings.

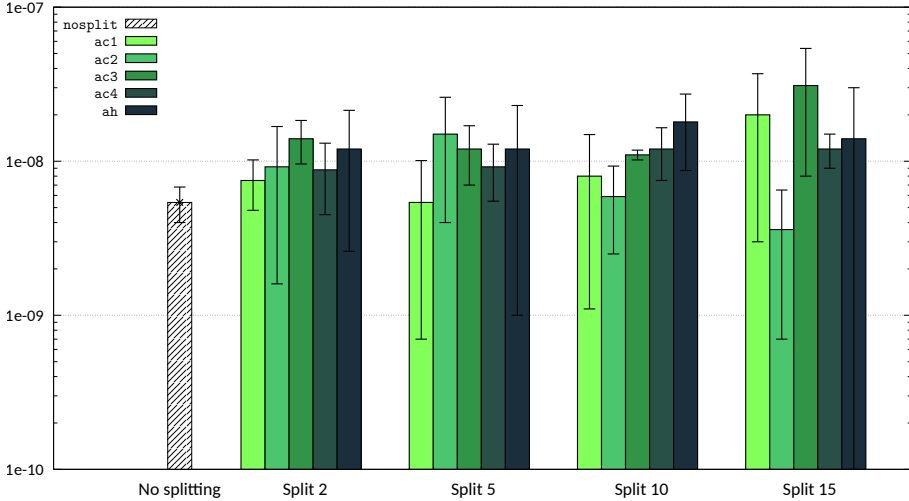


Figure 4.9: Exponential-failures oil pipeline; intervals precision for 3 h timeout

`nosplit` runs, are those where the event is less rare. This covers mostly the configurations where $k = 3$. Higher values of k allow a more fruitful layering of the state space and hence a more efficient application of multilevel splitting. This coincides with the analysis and results presented in [VA10], where RESTART with the importance function `ah` was employed.

Tables 4.4 and 4.5 show that the only runs where, for some execution settings, one or both experiments failed to produce a result, are precisely those where the event is most rare, viz. $n = 20$ and $k = 5$ for both failure distributions. This does not only make a proper analysis difficult, but also interferes with our attempts to corroborate the previous conjectures regarding the behaviour of multilevel splitting for higher values of k .

Hence, to allow a more robust and detailed analysis we replicated the experiments for the configuration $n = 20$ and $k = 5$ of the oil pipeline, for both exponential and Rayleigh failure times in the nodes. We let the simulations run for 3 h (wall time), using the precision of the intervals achieved as performance measure. Three independent experiments were run in Mendieta in this fashion; the results are presented in Figures 4.9 and 4.10. These values are the average of the precision of the intervals obtained from the three experiments run; the standard deviation is shown as whiskers on top of the bars.

To our surprise, even in these two configurations where the event is

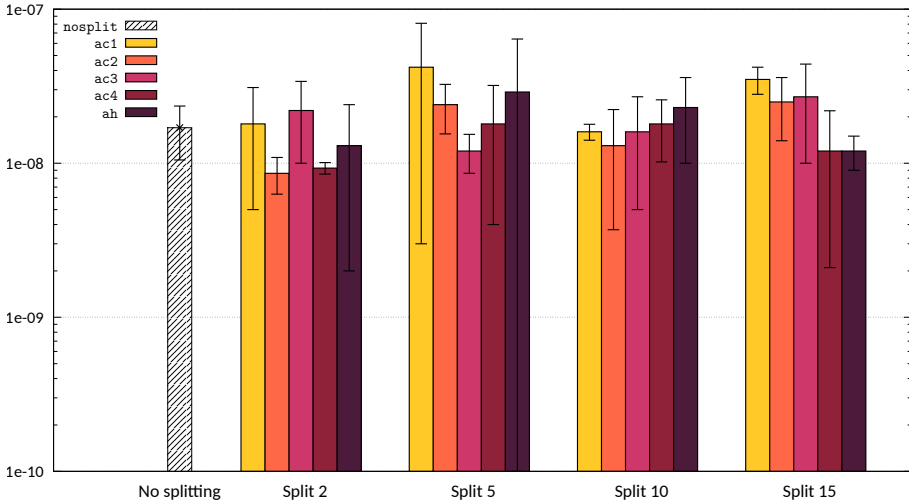


Figure 4.10: Rayleigh-failures oil pipeline; intervals precision for 3 h timeout

relatively rare, several splitting simulations were defeated by standard Monte Carlo. There are a few situations where particular RESTART settings clearly outperformed `nosplit`: e.g. in Figure 4.9 there is `ac2` for splitting 15; in Figure 4.10 there are `ac2` and `ac4` for splitting 2, `ac3` for splitting 5, and `ac4` and `ah` for splitting 15. However we would have expected a worse performance of `nosplit` w.r.t. the splitting variants.

Comparing these experiments for the different failure time distributions, we observe that the simulations which use splitting behaved worse (on average) for the exponentially distributed failures. Notice also how the standard deviation of most RESTART settings in Figures 4.9 and 4.10 is higher than that of standard Monte Carlo, and notice that such behaviour is more pronounced in the exponential (rather than the Rayleigh) variant. This last observation suggests a higher sensitivity to the seeding of the RNG by RESTART than by standard Monte Carlo, which would also be closely related to the splittings-thresholds fiasco. This also indicates that several simulations using splitting actually outperformed the ones employing `nosplit`, although the average behaviour appears to favour the latter, contrary to our initial expectations.

In a final attempt to better understand the behaviour of this oil pipeline model, we repeated the experiments for a higher wall time limit, namely 5 h. The hypothesis is that a longer execution could stabilise the behaviour of the

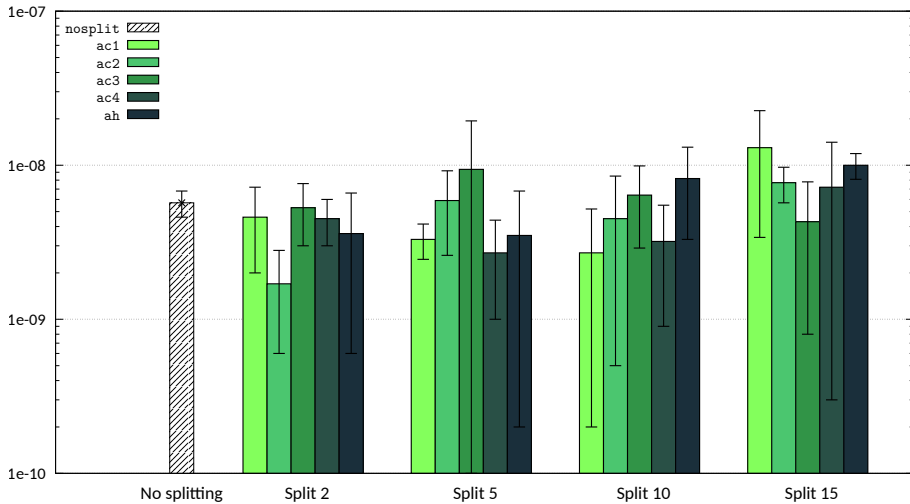


Figure 4.11: Exponential-failures oil pipeline; intervals precision for 5 h timeout

simulations in the long run. This should favour RESTART simulations over standard Monte Carlo, since a proper splitting should generate oversampling in an area rich in rare events, contrary to the single simulation approach of the `nosplit` setting.

The results of these last experiments, also run in Mendieta, are presented in Figures 4.11 and 4.12. Four instances were launched for each configuration and execution setting. All four succeeded in each case for the experiments with exponentially distributed failures in the nodes. However, in the Rayleigh cases and due to unavoidable issues with the hardware, only two or three runs for each case finished without external interruptions. The outcomes of the interrupted runs were discarded, and thus the samples used to compute the averages shown in Figure 4.12 consist in less than four values. That is why we consider the results from those experiments of lower quality than the ones presented in Figure 4.11.

On the one hand our conjectures regarding a better performance of RESTART were fulfilled in the exponential variant of the model, where most settings using splitting behaved (on average) better than standard Monte Carlo. We highlight results like `ac2` for splitting 2, `ac1` for splitting 5, and `ac4` for splittings 5 and 10, where the average interval width plus the standard deviation from measurements is still below the precision achieved by the `nosplit` simulations.

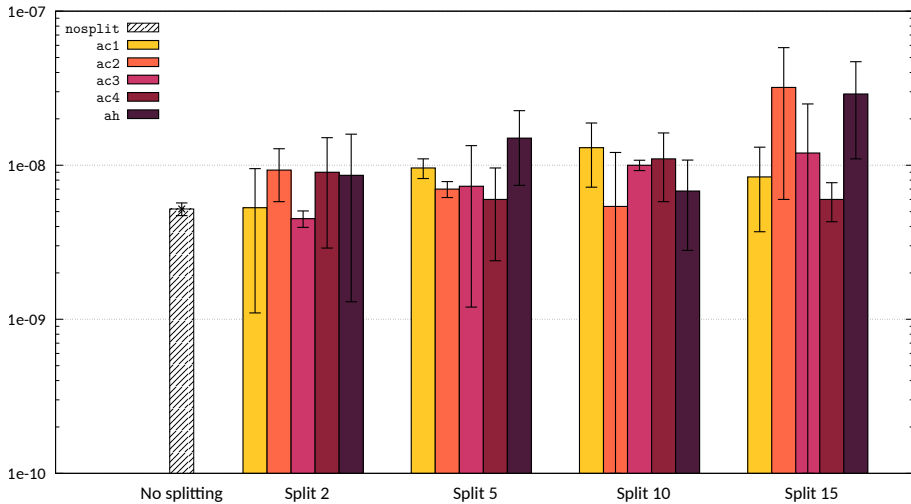


Figure 4.12: Rayleigh-failures oil pipeline; intervals precision for 5 h timeout

On the other hand the results for the oil pipeline with Rayleigh failure times are slightly disconcerting, because they show a tendency contrary to that of the exponential case, which we could predict successfully. That is, Figure 4.12 shows that simulations using splitting behaved worse than standard Monte Carlo in general, which is also at odds with the results previously presented in Figure 4.10 for the 3 h timeout.

Nonetheless, the matter can be settled by considering the following:

- the execution of the experiments was troublesome from the technical viewpoint, yielding smaller samples to compute the averages from;
- since there is evidence of a high sensitivity to the specific seed fed to the RNG, Figure 4.12 should be interpreted with care if we consider the previous item;
- as a matter of fact and in a more general sense, samples with more than—say—30 experimental runs should be used to reduce the standard deviation to reasonably small values;
- on top of all this we have the splittings-thresholds fiasco, revealed in the high variability observed for the different global splittings values, which complicates the comparison of any particular RESTART execution setting against standard Monte Carlo.

All this talks in favour of repeating the whole experimentation, running many more independent experiments per system configuration and execution setting, and maybe using longer execution times. However higher values of k should be studied first, since $k = 5$ may be simply not enough to observe a clear advantage of RESTART w.r.t. standard Monte Carlo. In particular [VA10] study the oil pipeline system for $k \in \{4, 6\}$, and report higher gains for the higher value of k . This will be the subject of future research.

To conclude this section, some comparisons are due among the different importance functions used in the RESTART runs. Neither Tables 4.3 to 4.5 nor Figures 4.9 to 4.12 suggest a clearly outstanding composition strategy outperforming the rest. Indeed, the fastest converging function varied much not only with the global splitting chosen, but also with the particular system configuration studied.

Still, the rich set of results presented along the section allows us to distill some useful information. Tables 4.4 and 4.5 show that `ac4` was either the best performing function or the runner up in most configurations where $n = 60$. Notice that in some settings it even outperformed the standard Monte Carlo approach, although the higher values of n favour the `nosplit` strategy as discussed. We suspect this is closely related to the large importance range offered by this function, higher than `ac1` and `ac3` for instance. Besides, since the function is derived from the specific property under study, `ac4` may fit the evolution of a simulation towards the rare event in a more natural way than e.g. functions `ac1` and `ac2`.

Moreover, `ac4` was among the most resilient to the change in splitting value, as it can be observed in Figures 4.9 to 4.12. In contrast we remark that `ac2`, which yielded quite good results for e.g. splitting 15 in Figure 4.9 and splitting 2 in Figure 4.11, showed a high variability related to the global splitting chosen. The simple summation implemented by `ac1` behaved better than expected, but was always outperformed by the functions implementing the ring/semiring composition strategies in most configurations—see for instance Tables 4.3 to 4.5.

The above indicates an overall very good behaviour of `ac4`, which in addition was never last in the ranking of importance functions for the most demanding configurations, i.e. the ones presented in Figures 4.9 to 4.12. We believe a more thorough study of the oil pipeline system, specifically testing higher values of k and other splitting values, would maintain this assertion and favour majorly `ac4`, whose good properties may be a result of the manner in which it is derived.

It seems clear there is much still to be learnt from this system. It is an interesting case study per se, due to its many applications in industry, but it also presents high potential for the application of importance splitting. In this our first approach we have seen that rarer regimes, where the system is more tolerant to failures due to high values of the parameter k , are beneficial to our techniques. Verifying the extent to which this holds with a more efficient version of our tool (which in particular has addressed the thresholds selection issue), is a challenge we intend to face in the near future.

Final remarks

In this thesis we have developed techniques to perform automated system model analysis by simulation in rare event regimes. We employ importance splitting (I-SPLIT) to steer the simulation of execution paths towards the rare event. We contributed with algorithms to derive the importance function on which I-SPLIT heavily relies. This way, the user input needed to run importance splitting does not differ from the usual input required by analyses which use standard Monte Carlo simulation, plus a global splitting value.

We divided our approach in two installments. Chapter 3 presents a first *monolithic approach* to build and store the importance function. The high quality of the resulting function was empirically verified in several case studies, but its requirement to expand the state space of the fully composed model is a major setback. Chapter 4 presents a second *compositional approach* which drops such requirement, at the expense of losing some insight into the global system semantics. However, choosing an adequate composition strategy guided by the rare event property can counter this sometimes. Furthermore, the composition strategy grants high flexibility to build a (global) importance function, with more potential than the monolithic approach.

Importance splitting is a complex technique. It requires articulating several decisions, e.g. the thresholds and the splitting for RESTART, in order to obtain some gain w.r.t. standard Monte Carlo. Besides developing an algorithmic basis, in this thesis we devise mechanisms to embed these algorithms in an automated application of I-SPLIT. The result, as desired, resembles the push-button approach from standard model checking.

Moreover we developed software tools (BLUEMOON and FIG) implementing our theory. This allowed us to validate our claims, running experiments on case studies taken from the RES literature. We compared the performance of standard Monte Carlo simulations against our automatic approach to I-SPLIT, showing the gain achieved by our proposal in rare event regimes. We also compared the performance of automatically built importance functions against functions chosen ad hoc for each model studied. The resulting outcomes witness that our proposal is quite versatile besides being automatic.

5.1 Future work

First and foremost, it is clear from the results and discussions presented in Sections 3.5 and 4.6 that our approach to select the importance thresholds for RESTART is far from optimal. We now suspect that the continuous-space hypothesis of both Adaptive Multilevel Splitting and Sequential Monte Carlo weights too heavily on the performance of these procedures. Adapting them to the discrete state space setting of Markov chains or IOSA yielded unsatisfactory results.

That is highly related to our strategy of choosing a global splitting value. This may yield optimal results in a continuous setting where thresholds can be chosen as close together as desired. However, in our experiments, having a single splitting value sometimes led to starvation, and sometimes to an overhead of offspring simulations, in spite of our efforts to counter this via the selection of the thresholds.

Late discussions with José Villén-Altamirano and Pedro R. D'Argenio have led us to believe that the global splitting strategy must be dropped, in order to obtain a (near-)optimal choice of thresholds. Instead one should select both the threshold and the splitting to perform upon reaching it, in an iterative procedure which evolves from the initial system state towards the rare event. An outline of an algorithm goes as follows[†]:

0. Initially regard every importance value of the current function as a potential threshold;
1. Launch n pilot RESTART simulations with global splitting 2;
2. Force an early termination if necessary, given one should spend less than 10% of the total computing budget in these *preliminary decisions*;
3. Approximate the probabilities $\{P_{i|0}\}$ from [VAVA06] with the quotient between: the number of simulations that reached the i -th importance value, and $2^i n$;
4. Approximate the probabilities $P_{i+1|1} = P_{i+1|0}P_{1|0}$ with the approximations of the previous item;

[†] José Villén-Altamirano proposed the original idea for steady-state analysis, later revised and updated in discussions with Arnd Hartmanns.

5. Using those values, compute the accumulated splitting coefficients $\{r_i\}$ from [VAVA06] using the equation $r_i = (P_{i|0}P_{i+1|1})^{-1/2} \in \mathbb{Q}$ also from that work;
6. Iteratively compute the (integral) splitting values $\{R_i\}$ for each potential threshold by means of the formula

$$R_i = \text{round} \left(\frac{r_i}{\prod_{j=1}^{i-1} R_j} \right);$$

7. If $R_i \leq 1$ then the i -th importance value is not a threshold;
8. Else it is, and simulations reaching it should spawn $R_i \in \mathbb{N}$ offsprings.

Many other potential improvements on the results from this thesis reside on the algorithms which derive the importance function. Their proven effectivity notwithstanding, myriads of variations can be tested, perhaps on extended versions of IOSA, or also on different modelling formalisms.

The first change that might come into mind is considering the probabilistic weights of transitions in an adaptation of Algorithm 1. Notice this cannot always be performed, e.g. Markov chains are generally represented with abstract data types that would allow it, but the stochastic component of IOSA (as presented here) lies further from reach, “hidden in the clocks.”

Another extension to our compositional approach in particular, is developing other automatic ways to compose the local importance functions. We found that a DNF expression of the rare event is both natural and useful to derive a composition strategy. However, more complex yet structured ways to express the property query could be considered, maintaining the capacity to distill a composition strategy from them. In that sense we are currently drawing our attention towards the theory of repairable Dynamic Fault Trees—see [RS15] and references therein.

In a more distant appreciation, this thesis builds the importance function mostly based on the structure of the model. The rare event property is a key component as well, and even more so for the compositional approach, but the distance between values upon which the importance function is based is imprinted by the adjacency graph of the system model. A different approach would be to reverse this strategy, starting to build the function from the property expression and modifying it as one traverses the model, as Sedwards et al. do in [JLS13, JLST15]. In this direction we believe that the *counting fluents* of [RDDA15] could provide a richer framework.

Last but not least, one of the main motivations of this thesis is the development of software tools to offer off-the-shelf implementations of our proposals. In that respect BLUEMOON was devised mostly as a prototype to test the validity of our strategies, whereas the design and implementation of FIG are planned on a vaster scale.

It would be very interesting to see further development of the FIG tool. On the one hand there are several efficiency boosts close at hand, like a trivial parallelization of the interval update mechanisms, which could improve the performance of estimations at very low coding effort. On the other hand there are deeper issues, stemming from the algorithmic choices of the tool, which also affect the general performance. The thresholds selection mechanism and the deterministic truncation of unpromising paths are examples of these. Studying the effect of a change in such algorithms sounds promising, mostly the one used to choose the thresholds as earlier mentioned.

Furthermore, FIG currently implements a single importance splitting algorithm. Even though this algorithm (RESTART) was chosen due to its fine general properties, there are no singular bounds between the tool and RESTART. Recall that the importance function can be used as a black box by most importance splitting strategies. That, plus the modular design of FIG, should make the addition of further *simulation engines* (like Fixed Effort, see Section 2.5.2) quite a straightforward task.

Appendix:

System models

A

This appendix presents the source code of all the models used to produce the results included in this thesis. Two modelling formalisms are used: all systems studied in Chapter 3 are modelled in the PRISM input language; the ones studied in Chapter 4 are written using the IOSA model syntax instead.

A.1 Tandem queue

PRISM model of a continuous time tandem queue, used to produce the results presented in Section 3.5.2.

```
1  ctmc
2
3  const int c;           // Queues capacity
4  const double lambda = 3; // rate(-> q1      )
5  const double mu1 = 2;  // rate(   q1 -> q2   )
6  const double mu2 = 6;  // rate(           q2 ->)
7  // Values taken from Marnix Garvels' Ph.D. Thesis:
8  // The splitting method in rare event simulation, p. 85.
9
10 module ContinuousTandemQueue
11
12     q1: [0..c-1] init 0;
13     q2: [0..c-1] init 1;
14     arr: [0..2] init 0; // Arrival: (0:none) (1:lost) (2:successful)
15     lost: [0..1] init 0; // Package loss in q2: (0:none) (1:lost)
16
17     // Package arrival at first queue
18     [] q1<c-1 -> lambda: (arr'=2) & (lost'=0) & (q1'=q1+1);
19     [] q1=c-1 -> lambda: (arr'=1) & (lost'=0);
20
21     // Passing from first to second queue
22     [] q1>0 & q2<c-1 -> mu1: (arr'=0) & (lost'=0) & (q1'=q1-1) & (q2'=q2+1);
23
24     [] q1>0 & q2=c-1 -> mu1: (arr'=0) & (lost'=1) & (q1'=q1-1);
25
26     // Package departure from second queue
27     [] q2>0 -> mu2: (arr'=0) & (lost'=0) & (q2'=q2-1);
```

```

28 endmodule
29
30 label "goal" = lost=1;
31 label "stop" = q2=0;
32 label "running" = q2!=0;
33 label "reference" = true;

```

A.2 Discrete time tandem queue

PRISM model of a discrete time tandem queue, used to produce the results presented in Section 3.5.3.

```

1 dtmc
2
3 const int c;           // Queues capacity
4 const double parr = 0.1; // Prob(-> q1      )
5 const double ps1 = 0.14; // Prob(      q1 -> q2  )
6 const double ps2 = 0.19; // Prob(                q2 ->)
7
8 module DiscreteTandemQueue
9
10     q1: [0..c] init 0;
11     q2: [0..c] init 0;
12     arr1: [0..2] init 0; // Arrival: (0:none) (1:lost) (2:successful)
13     lost2: [0..1] init 0; // Package loss in q2: (0:none) (1:lost)
14
15     [] (q1=0) & (q2=0)
16         -> (parr): (q1'=q1+1) & (arr1'=2) & (lost2'=0)
17             + (1-parr): (arr1'=0) & (lost2'=0);
18
19     [] (0<q1 & q1<c) & (q2=0)
20         -> (parr*ps1): (q2'=q2+1) & (arr1'=2) & (lost2'=0)
21             + (parr*(1-ps1)): (q1'=q1+1) & (arr1'=2) & (lost2'=0)
22             + (ps1*(1-parr)): (q1'=q1-1) & (q2'=q2+1) & (arr1'=0) & (lost2'=0)
23
24             + ((1-parr)*(1-ps1)): (arr1'=0) & (lost2'=0);
25
26     [] (q1=c) & (q2=0)
27         -> (parr*ps1): (q2'=q2+1) & (arr1'=2) & (lost2'=0)
28             + (parr*(1-ps1)): (arr1'=1) & (lost2'=0)
29             + ((1-parr)*ps1): (q1'=q1-1) & (q2'=q2+1) & (arr1'=0) & (lost2'=0)
30
31             + ((1-parr)*(1-ps1)): (arr1'=0) & (lost2'=0);
32
33     [] (q1=0) & (0<q2)
34         -> (parr*ps2): (q1'=q1+1) & (q2'=q2-1) & (arr1'=2) & (lost2'=0)
35             + (parr*(1-ps2)): (q1'=q1+1) & (arr1'=2) & (lost2'=0)
36             + ((1-parr)*ps2): (q2'=q2-1) & (arr1'=0) & (lost2'=0)
37             + ((1-parr)*(1-ps2)): (arr1'=0) & (lost2'=0);

```

```

36
37 [] (0<q1 & q1<c) & (0<q2 & q2<c)
38   -> (parr*ps1*ps2): (arr1'=2) & (lost2'=0)
39     + (parr*ps1*(1-ps2)): (q2'=q2+1) & (arr1'=2) & (lost2'=0)
40     + (parr*(1-ps1)*ps2): (q1'=q1+1) & (q2'=q2-1) & (arr1'=2)
41                                   & (lost2'=0)
42     + (parr*(1-ps1)*(1-ps2)): (q1'=q1+1) & (arr1'=2) & (lost2'=0)
43     + ((1-parr)*ps1*ps2): (q1'=q1-1) & (arr1'=0) & (lost2'=0)
44     + ((1-parr)*ps1*(1-ps2)): (q1'=q1-1) & (q2'=q2+1) & (arr1'=0)
45                                   & (lost2'=0)
46     + ((1-parr)*(1-ps1)*ps2): (q2'=q2-1) & (arr1'=0) & (lost2'=0)
47     + ((1-parr)*(1-ps1)*(1-ps2)): (arr1'=0) & (lost2'=0);
48
49 [] (q1=c) & (0<q2 & q2<c)
50   -> (parr*ps1*ps2): (arr1'=2) & (lost2'=0)
51     + (parr*ps1*(1-ps2)): (q2'=q2+1) & (arr1'=2) & (lost2'=0)
52     + (parr*(1-ps1)*ps2): (q2'=q2-1) & (arr1'=1) & (lost2'=0)
53     + (parr*(1-ps1)*(1-ps2)): (arr1'=1) & (lost2'=0)
54     + ((1-parr)*ps1*ps2): (q1'=q1-1) & (arr1'=0) & (lost2'=0)
55     + ((1-parr)*ps1*(1-ps2)): (q1'=q1-1) & (q2'=q2+1) & (arr1'=0)
56                                   & (lost2'=0)
57     + ((1-parr)*(1-ps1)*ps2): (q2'=q2-1) & (arr1'=0) & (lost2'=0)
58     + ((1-parr)*(1-ps1)*(1-ps2)): (arr1'=0) & (lost2'=0);
59
60 [] (0<q1 & q1<c) & (q2=c)
61   -> (parr*ps1*ps2): (arr1'=2) & (lost2'=0)
62     + (parr*ps1*(1-ps2)): (arr1'=2) & (lost2'=1)
63     + (parr*(1-ps1)*ps2): (q1'=q1+1) & (q2'=q2-1) & (arr1'=2)
64                                   & (lost2'=0)
65     + (parr*(1-ps1)*(1-ps2)): (q1'=q1+1) & (arr1'=2) & (lost2'=0)
66     + ((1-parr)*ps1*ps2): (q1'=q1-1) & (arr1'=0) & (lost2'=0)
67     + ((1-parr)*ps1*(1-ps2)): (q1'=q1-1) & (arr1'=0) & (lost2'=1)
68     + ((1-parr)*(1-ps1)*ps2): (q2'=q2-1) & (arr1'=0) & (lost2'=0)
69     + ((1-parr)*(1-ps1)*(1-ps2)): (arr1'=0) & (lost2'=0);
70
71 [] (q1=c) & (q2=c)
72   -> (parr*ps1*ps2): (arr1'=2) & (lost2'=0)
73     + (parr*ps1*(1-ps2)): (arr1'=2) & (lost2'=1)
74     + (parr*(1-ps1)*ps2): (q2'=q2-1) & (arr1'=1) & (lost2'=0)
75     + (parr*(1-ps1)*(1-ps2)): (arr1'=1) & (lost2'=0)
76     + ((1-parr)*ps1*ps2): (q1'=q1-1) & (arr1'=0) & (lost2'=0)
77     + ((1-parr)*ps1*(1-ps2)): (q1'=q1-1) & (arr1'=0) & (lost2'=1)
78     + ((1-parr)*(1-ps1)*ps2): (q2'=q2-1) & (arr1'=0) & (lost2'=0)
79     + ((1-parr)*(1-ps1)*(1-ps2)): (arr1'=0) & (lost2'=0);
80 endmodule
81
82 label "goal" = lost2=1;
83 label "reference" = true; // arr1!=0;

```

A.3 Mixed open/closed queue

PRISM model of a mixed open/closed queue, used to produce the results presented in Section 3.5.4.

```

1  ctmc
2
3  const int b;           // Open queue (oq) capacity
4  const int N2 = 1;     // Closed system (cq+cqq) fixed size
5  const double l = 1;   // oq arrival rate
6  const double m11 = 4; // oq Server1 rate
7  const double m12 = 2; // cq Server1 rate
8  const double m2;     // cq Server2 rate
9  // Values taken from Glasserman, Heidelberger, Shahabuddin, and Zajic:
10 // Multilevel Splitting For Estimating Rare Event Probabilities,
11 // Operations Research, Vol. 47, No. 4, July-August 1999, pp. 585-600
12
13 // System queues
14 global oq: [0..b] init 0; // Open queue
15 global cq: [0..N2] init 0; // Closed queue
16
17 module Arrival
18     lost: bool init false;
19     [] oq<b-1 -> l: (oq'=oq+1);
20     [] oq=b-1 -> l: (lost'=true);
21 endmodule
22
23 module Server1
24     reset: bool init false;
25     [] oq>1 & cq=0 -> m11: (oq'=oq-1);
26     [] oq=1 & cq=0 -> m11: (reset'=true);
27     [] cq>1 -> m12: (cq'=cq-1);
28     [] cq=1 & oq>0 -> m12: (cq'=cq-1);
29     [] cq=1 & oq=0 -> m12: (reset'=true);
30 endmodule
31
32 module Server2
33     [] cq<N2 -> m2: (cq'=cq+1);
34 endmodule
35
36 label "goal" = lost;
37 label "stop" = reset;
38 label "running" = !reset;

```

A.4 Queuing system with breakdowns

PRISM model of a queue with breakdowns, used to produce the results presented in Section 3.5.5.

```

1  ctmc
2
3  // The following values were extracted from Kroese & Nicola:
4  // Efficient estimation of overflow probabilities in queues
5  // with breakdowns, Performance Evaluation, 36-37, 1999, pp. 471-484.
6  // This model corresponds to the system described in the section 4.4
7  // (p. 481) of said article.
8
9  // Buffer capacity
10 const int k;
11
12 // Server
13 const double mu = 100;
14 const double xi = 3;
15 const double delta = 4;
16
17 // Sources of Type 1
18 const int NSrc1 = 5;
19 const double lambda1 = 3;
20 const double alpha1 = 3;
21 const double beta1 = 2;
22
23 // Sources of Type 2
24 const int NSrc2 = 5;
25 const double lambda2 = 6;
26 const double alpha2 = 1;
27 const double beta2 = 4;
28
29 module QueueWithBreakdowns
30
31   // Initializations
32   lost: bool init false;
33   reset: bool init false;
34   buf: [0..K-1] init 1; // Buffer, initially with one customer
35   server: bool init false; // Server, initially down
36   src1: [0..NSrc1] init 0; // Sources of Type 1, initially none active
37   src2: [0..NSrc2] init 1; // Sources of Type 2, initially one active
38
39   // Sources failure and recovery
40   [] src1>0 -> (src1 * beta1) : (src1'=src1-1);
41   [] src1<NSrc1 -> ((NSrc1-src1) * alpha1) : (src1'=src1+1);
42   [] src2>0 -> (src2 * beta2) : (src2'=src2-1);
43   [] src2<NSrc2 -> ((NSrc2-src2) * alpha2) : (src2'=src2+1);
44
45   // Server failure and recovery
46   [] server -> xi: (server'=false);
47   [] !server -> delta: (server'=true);
48
49   // Buffer in
50   [] src1>0 & buf<K-1 -> (src1 * lambda1) : (buf'=buf+1);
51   [] src1>0 & buf=K-1 -> (src1 * lambda1) : (lost'=true);
52   [] src2>0 & buf<K-1 -> (src2 * lambda2) : (buf'=buf+1);
53   [] src2>0 & buf=K-1 -> (src2 * lambda2) : (lost'=true);

```



```

54
55     // Buffer out
56     [] server & buf>1 -> mu : (buf'=buf-1);
57     [] server & buf=1 -> mu : (reset'=true);
58 endmodule
59
60 label "goal" = lost;
61 label "stop" = reset;
62 label "running" = !reset;

```

A.5 Database system with redundancy

PRISM model of a database system with redundancy, used in Example 7.

```

1  ctmc
2
3  // The following values were extracted from José Villén-Altamirano,
4  // Importance functions for RESTART simulation of highly-dependable
5  // systems, Simulation, Vol. 83, Issue 12, December 2007, pp. 821-828.
6
7  // Redundancy level, viz. how many breaks produce a system failure
8  const int RED;
9
10 // Processors
11 global P1: [0..RED] init RED;
12 global P2: [0..RED] init RED;
13 const double PF = 2000; // Processors' mean time to failure (in hours)
14 const double IPF = 0.01; // Processors' inter-type failure rate
15
16 // Controllers
17 global C1: [0..RED] init RED;
18 global C2: [0..RED] init RED;
19 const double CF = 2000; // Controllers' mean time to failure (in hours)
20
21 // Disk clusters
22 global D1: [0..RED+2] init RED+2;
23 global D2: [0..RED+2] init RED+2;
24 global D3: [0..RED+2] init RED+2;
25 global D4: [0..RED+2] init RED+2;
26 global D5: [0..RED+2] init RED+2;
27 global D6: [0..RED+2] init RED+2;
28 const double DF = 6000; // Disks' mean time to failure (in hours)
29
30 // Repair rates for failures of type 1 and 2 resp.
31 const double R1 = 1.0;
32 const double R2 = 0.5;
33
34 module Processors
35     [] P1 > 0 -> (P1/PF)*(1-IPF): (P1'=P1-1)
36         + (P1/PF)*( IPF): (P1'=P1-1)&(P2'=P2-1);

```

```

37     [] P2 > 0 -> (P2/PF)*(1-IPF): (P2'=P2-1)
38                 + (P2/PF)*( IPF): (P2'=P2-1)&(P1'=P1-1);
39 endmodule
40
41 module Controllers
42     [] C1>0 -> C1/CF: (C1'=C1-1);
43     [] C2>0 -> C2/CF: (C2'=C2-1);
44 endmodule
45
46 module DiskClusters
47     [] D1>0 -> D1/DF: (D1'=D1-1);
48     [] D2>0 -> D2/DF: (D2'=D2-1);
49     [] D3>0 -> D3/DF: (D3'=D3-1);
50     [] D4>0 -> D4/DF: (D4'=D4-1);
51     [] D5>0 -> D5/DF: (D5'=D5-1);
52     [] D6>0 -> D6/DF: (D6'=D6-1);
53 endmodule
54
55 // Number of failed components in the system
56 formula NFails = (2*RED-P1-P2)
57                 + (2*RED-C1-C2)
58                 + (6*(RED+2)-D1-D2-D3-D4-D5-D6);
59
60 // Operational Components in the minimal cutset
61 formula minOC = min(P1, P2,
62                    C1, C2,
63                    D1-2, D2-2, D3-2, D4-2, D5-2, D6-2);
64
65 module Repairman
66     f: bool init false;
67     // Type 1 failures on processors ...
68     [] !f & P1<RED -> 0.5 * R1 * (RED-P1)/NFails: (P1'=P1+1)
69                 + 0.5 * R1 * (RED-P1)/NFails: (P1'=P1+1) & (f'=!f);
70     [] !f & P2<RED -> 0.5 * R1 * (RED-P2)/NFails: (P2'=P2+1)
71                 + 0.5 * R1 * (RED-P2)/NFails: (P2'=P2+1) & (f'=!f);
72     // ... on controllers ...
73     [] !f & C1<RED -> 0.5 * R1 * (RED-C1)/NFails: (C1'=C1+1)
74                 + 0.5 * R1 * (RED-C1)/NFails: (C1'=C1+1) & (f'=!f);
75     [] !f & C2<RED -> 0.5 * R1 * (RED-C2)/NFails: (C2'=C2+1)
76                 + 0.5 * R1 * (RED-C2)/NFails: (C2'=C2+1) & (f'=!f);
77     // ... and on disks.
78     [] !f & D1<RED+2 -> 0.5 * R1 * (RED+2-D1)/NFails: (D1'=D1+1)
79                 + 0.5 * R1 * (RED+2-D1)/NFails: (D1'=D1+1) & (f'=!f);
80
81     [] !f & D2<RED+2 -> 0.5 * R1 * (RED+2-D2)/NFails: (D2'=D2+1)
82                 + 0.5 * R1 * (RED+2-D2)/NFails: (D2'=D2+1) & (f'=!f);
83
84     [] !f & D3<RED+2 -> 0.5 * R1 * (RED+2-D3)/NFails: (D3'=D3+1)
85                 + 0.5 * R1 * (RED+2-D3)/NFails: (D3'=D3+1) & (f'=!f);
86
87     [] !f & D4<RED+2 -> 0.5 * R1 * (RED+2-D4)/NFails: (D4'=D4+1)
88                 + 0.5 * R1 * (RED+2-D4)/NFails: (D4'=D4+1) & (f'=!f);
89
90     [] !f & D5<RED+2 -> 0.5 * R1 * (RED+2-D5)/NFails: (D5'=D5+1)
91                 + 0.5 * R1 * (RED+2-D5)/NFails: (D5'=D5+1) & (f'=!f);
92
93     [] !f & D6<RED+2 -> 0.5 * R1 * (RED+2-D6)/NFails: (D6'=D6+1)
94                 + 0.5 * R1 * (RED+2-D6)/NFails: (D6'=D6+1) & (f'=!f);

```

```

86     [] !f & D5<RED+2 -> 0.5 * R1 * (RED+2-D5)/NFails: (D5'=D5+1)
87         + 0.5 * R1 * (RED+2-D5)/NFails: (D5'=D5+1) & (f'=!f);

88     [] !f & D6<RED+2 -> 0.5 * R1 * (RED+2-D6)/NFails: (D6'=D6+1)
89         + 0.5 * R1 * (RED+2-D6)/NFails: (D6'=D6+1) & (f'=!f);

90     // Type 2 failures on processors ...
91     [] f & P1<RED -> 0.5 * R2 * (RED-P1)/NFails: (P1'=P1+1)
92         + 0.5 * R2 * (RED-P1)/NFails: (P1'=P1+1) & (f'=!f);
93     [] f & P2<RED -> 0.5 * R2 * (RED-P2)/NFails: (P2'=P2+1)
94         + 0.5 * R2 * (RED-P2)/NFails: (P2'=P2+1) & (f'=!f);
95     // ... on controllers ...
96     [] f & C1<RED -> 0.5 * R2 * (RED-C1)/NFails: (C1'=C1+1)
97         + 0.5 * R2 * (RED-C1)/NFails: (C1'=C1+1) & (f'=!f);
98     [] f & C2<RED -> 0.5 * R2 * (RED-C2)/NFails: (C2'=C2+1)
99         + 0.5 * R2 * (RED-C2)/NFails: (C2'=C2+1) & (f'=!f);
100    // ... and on disks.
101    [] f & D1<RED+2 -> 0.5 * R2 * (RED+2-D1)/NFails: (D1'=D1+1)
102        + 0.5 * R2 * (RED+2-D1)/NFails: (D1'=D1+1) & (f'=!f);

103    [] f & D2<RED+2 -> 0.5 * R2 * (RED+2-D2)/NFails: (D2'=D2+1)
104        + 0.5 * R2 * (RED+2-D2)/NFails: (D2'=D2+1) & (f'=!f);

105    [] f & D3<RED+2 -> 0.5 * R2 * (RED+2-D3)/NFails: (D3'=D3+1)
106        + 0.5 * R2 * (RED+2-D3)/NFails: (D3'=D3+1) & (f'=!f);

107    [] f & D4<RED+2 -> 0.5 * R2 * (RED+2-D4)/NFails: (D4'=D4+1)
108        + 0.5 * R2 * (RED+2-D4)/NFails: (D4'=D4+1) & (f'=!f);

109    [] f & D5<RED+2 -> 0.5 * R2 * (RED+2-D5)/NFails: (D5'=D5+1)
110        + 0.5 * R2 * (RED+2-D5)/NFails: (D5'=D5+1) & (f'=!f);

111    [] f & D6<RED+2 -> 0.5 * R2 * (RED+2-D6)/NFails: (D6'=D6+1)
112        + 0.5 * R2 * (RED+2-D6)/NFails: (D6'=D6+1) & (f'=!f);

113    endmodule
114
115    label "reference" = true;
116    label "goal" = (P1=0) | (P2=0)
117                | (C1=0) | (C2=0)
118                | (D1<=2) | (D2<=2) | (D3<=2) | (D4<=2) | (D5<=2) | (D6<=2);

```

A.6 Tandem queue

IOSA model of a (continuous time) tandem queue, used to produce the results presented in Section 4.6.2.

```

1  const int c = 8; // Capacity of both queues
2

```

```

3 // The following values were taken from Marnix Garvels' PhD Thesis:
4 // The splitting method in rare event simulation, p. 85.
5 const int lambda = 3; // rate(-> q1      )
6 const int mu1 = 2; // rate(  q1 -> q2  )
7 const int mu2 = 6; // rate(          q2 ->)
8
9 // The following values are in p. 61 of the same work:
10 // const int lambda = 1;
11 // const int mu1 = 4;
12 // const int mu2 = 2;
13
14 module Arrivals
15     clk0: cclock; // External arrivals ~ Exponential(lambda)
16     [P0!] @ clk0 -> (clk0' = exponential(lambda));
17 endmodule
18
19 module Queue1
20     q1: [0..c];
21     clk1: cclock; // Queue1 processing ~ Exponential(mu1)
22     // Packet arrival
23     [P0?] q1 == 0      -> (q1' = q1+1) & (clk1' = exponential(mu1));
24     [P0?] q1 > 0 & q1 < c -> (q1' = q1+1);
25     [P0?] q1 == c      -> ;
26     // Packet processing
27     [P1!] q1 == 1 @ clk1 -> (q1' = q1-1);
28     [P1!] q1 > 1 @ clk1 -> (q1' = q1-1) & (clk1' = exponential(mu1));
29 endmodule
30
31 module Queue2
32     q2: [0..c] init 1;
33     clk2: cclock; // Queue2 processing ~ Exponential(mu2)
34     // Packet arrival
35     [P1?] q2 == 0      -> (q2' = q2+1) & (clk2' = exponential(mu2));
36     [P1?] q2 > 0 & q2 < c -> (q2' = q2+1);
37     [P1?] q2 == c      -> ;
38     // Packet processing
39     [P2!] q2 == 1 @ clk2 -> (q2' = q2-1);
40     [P2!] q2 > 1 @ clk2 -> (q2' = q2-1) & (clk2' = exponential(mu2));
41 endmodule
42
43 properties
44     P( q2 > 0 U q2 == c ) // transient
45     S( q2 == c ) // steady-state
46 endproperties

```

A.7 Tandem queue (alternative)

PRISM model of a (continuous time) tandem queue, used to produce the results presented in Section 4.6.2. This queue, modelled in the PRISM input

language, is equivalent to the one described using the IOSA model syntax in Appendix A.6.

```

1  ctmc
2
3  const int c;           // Queues capacity
4  const double lambda = 3; // rate(--> q1      )
5  const double mu1 = 2;  // rate(   q1 --> q2  )
6  const double mu2 = 6;  // rate(           q2 -->)
7
8  module Arrival
9    // External packet arrival
10   [P0] true -> lambda: true;
11 endmodule
12
13 module Queue1
14   q1: [0..c] init 0;
15   // Packet arrival
16   [P0] q1<c -> 1: (q1'=q1+1);
17   [P0] q1=c -> 1: true;
18   // Packet processing
19   [P1] q1>0 -> mu1: (q1'=q1-1);
20 endmodule
21
22 module Queue2
23   q2: [0..c] init 1;
24   // Packet arrival
25   [P1] q2<c -> 1: (q2'=q2+1);
26   [P1] q2=c -> 1: true;
27   // Packet processing
28   [P2] q2>0 -> mu2: (q2'=q2-1);
29 endmodule

```

A.8 Triple tandem queue

IOSA model of a non-Markovian triple tandem queue, used to produce the results presented in Section 4.6.3.

```

1  // The following values were extracted from José Villén-Altamirano,
2  // RESTART simulation of networks of queues with Erlang service times,
3  // Winter Simulation Conference, 2009, pp. 1146-1154.
4  // This model corresponds to the system described in Section 4.1
5
6  const int a = 2;      // Service time shape parameter ('alpha', all queues)
7  const int b1 = 3;   // Service time scale parameter ('beta1', Queue1)
8  const int b2 = 4;   // Service time scale parameter ('beta2', Queue2)
9  const int b3 = 6;   // Service time scale parameter ('beta3', Queue3)
10 const int L = 7;    // Threshold occupancy (Queue3)
11 const int c = L+5;  // Queues capacity (all queues)

```

```

12
13 // Combinations tested in J. V-A's article:
14 //      L alpha beta1 beta2 beta3
15 // A) 18  2   3   4   6
16 // B) 13  3   4.5 6   9
17 // C) 20  2   6   4   6
18 // D) 16  3   9   6   9
19 // E) 24  2  10   8   6
20 // F) 21  3  15  12   9
21 //
22 // Those values of 'L' yield rare events of probability ~ 10-15.
23 // Alternatively the following values yield rare events ~ 10-9:
24 // L = (A:11, B:7, C:11, D:9, E:14, F:12)
25
26 module Arrivals
27     clk0: cclock; // External arrivals ~ Exponential(1)
28     [P0!] @ clk0 -> (clk0' = exponential(1));
29 endmodule
30
31 module Queue1
32     q1: [0..c];
33     clk1: cclock; // Queue1 processing ~ Erlang(alpha;beta1)
34     // Packet arrival
35     [P0?] q1 == 0          -> (q1' = q1+1) & (clk1' = erlang(a,b1));
36     [P0?] q1 > 0 & q1 < c -> (q1' = q1+1);
37     [P0?] q1 == c         -> ;
38     // Packet processing
39     [P1!] q1 == 1 @ clk1 -> (q1' = q1-1);
40     [P1!] q1 > 1 @ clk1 -> (q1' = q1-1) & (clk1' = erlang(a,b1));
41 endmodule
42
43 module Queue2
44     q2: [0..c];
45     clk2: cclock; // Queue2 processing ~ Erlang(alpha;beta2)
46     // Packet arrival
47     [P1?] q2 == 0          -> (q2' = q2+1) & (clk2' = erlang(a,b2));
48     [P1?] q2 > 0 & q2 < c -> (q2' = q2+1);
49     [P1?] q2 == c         -> ;
50     // Packet processing
51     [P2!] q2 == 1 @ clk2 -> (q2' = q2-1);
52     [P2!] q2 > 1 @ clk2 -> (q2' = q2-1) & (clk2' = erlang(a,b2));
53 endmodule
54
55 module Queue3
56     q3: [0..c];
57     clk3: cclock; // Queue3 processing ~ Erlang(alpha;beta3)
58     // Packet arrival
59     [P2?] q3 == 0          -> (q3' = q3+1) & (clk3' = erlang(a,b3));
60     [P2?] q3 > 0 & q3 < c -> (q3' = q3+1);
61     [P2?] q3 == c         -> ;
62     // Packet processing
63     [P3!] q3 == 1 @ clk3 -> (q3' = q3-1);
64     [P3!] q3 > 1 @ clk3 -> (q3' = q3-1) & (clk3' = erlang(a,b3));

```

```

65 endmodule
66
67 properties
68     S( q3 >= L ) // steady-state
69 endproperties

```

A.9 Queuing system with breakdowns

Summarised version of the IOSA model for a queue with breakdowns, used to produce the results presented in Section 4.6.4.

```

1 // The following values were extracted from Kroese & Nicola,
2 // Efficient estimation of overflow probabilities in queues
3 // with breakdowns, Performance Evaluation, 36-37, 1999, pp. 471-484.
4 // This model corresponds to the system described in Section 4.4
5
6 // Sources of Type 1
7 const int lambda1 = 3; // Production rate
8 const int alpha1  = 3; // Repair rate
9 const int beta1   = 2; // Fail rate
10
11 // Sources of Type 2
12 const int lambda2 = 6; // Production rate
13 const int alpha2  = 1; // Repair rate
14 const int beta2   = 4; // Fail rate
15
16 // Server
17 const int mu = 100; // Processing rate
18 const int delta = 4; // Repair rate
19 const int gama = 3; // Fail rate
20
21 // Buffer capacity: 40, 60, 80, 100, 120, 140, 160
22 const int K = 120;
23
24 ///////////////////////////////////////////////////////////////////
25 //
26 // Type 1 Sources | Total: 5
27 // | Initially on: 0
28
29 module T1S1
30     on11: bool init false;
31     clkF11: clock; // Type 1 sources Failures ~ exp(beta1)
32     clkR11: clock; // Type 1 sources Repairs ~ exp(alpha1)
33     clkP11: clock; // Type 1 sources Production ~ exp(lambda1)
34     // Breakdowns
35     [] on11 @ clkF11 -> (on11' = false) &
36         (clkR11' = exponential(alpha1));
37     [] !on11 @ clkR11 -> (on11' = true) &
38         (clkF11' = exponential(beta1)) &

```



```

213                                     (clkP25'= exponential(lambda2));
214     // Production
215     [p25!] on25 @ clkP25 -> (clkP25'= exponential(lambda2));
216 endmodule
217
218 ///////////////////////////////////////////////////////////////////
219 //
220 // Buffered server | Keeps track of 'overflow' and 'reset'
221 // | Translated from bluemoon's homonymous model
222
223 module BufferedServer
224     buf: [0..K] init 1;
225     clkF: clock; // Server Failure ~ exp(gama)
226     clkR: clock; // Server Repair ~ exp(delta)
227     clkP: clock; // Server Processing ~ exp(mu)
228     on: bool init false; // Server on?
229     reset: bool init false;
230     // Server failure and recovery
231     [] on @ clkF -> (on'= false) &
232                 (clkR'= exponential(delta));
233     [] !on @ clkR -> (on'= true) &
234                 (clkF'= exponential(gama)) &
235                 (clkP'= exponential(mu));
236     // Buffer out (dequeuing by server processing)
237     [] on & buf > 1 @ clkP -> (buf'= buf-1) &
238                             (clkP'= exponential(mu));
239     [] on & buf == 1 @ clkP -> (buf'= buf-1) &
240                             (reset'= true);
241     // Buffer in (enqueueing by sources production)
242     [p11?] buf == 0 -> (buf'= buf+1) & (clkP'= exponential(mu));
243     [p11?] 0 < buf & buf < K -> (buf'= buf+1);
244     [p11?] buf == K -> ;
245     :
277     [p25?] buf == 0 -> (buf'= buf+1) & (clkP'= exponential(mu));
278     [p25?] 0 < buf & buf < K -> (buf'= buf+1);
279     [p25?] buf == K -> ;
280 endmodule
281
282 properties
283     P( !reset U buf == K ) // transient
284 endproperties

```

A.10 Database system with redundancy

Summarised version of an IOSA model for a database with redundancy 2. These models were used to produce the results presented in Section 4.6.5. The number of system components increases with the redundancy value.


```

605     [] p11f & p11t==2 @ p11cR2 -> (p11f'= false) &
606                                     (p11cF1'= exponential(1/(PF*2))) &
607                                     (p11cF2'= exponential(1/(PF*2)));
608 endmodule
        :
672 properties
673     S( (d11f & d12f) | (d11f & d13f) | (d11f & d14f) | // Disk cl. #1
674         (d12f & d13f) | (d12f & d14f) | (d13f & d14f) |
675         (d21f & d22f) | (d21f & d23f) | (d21f & d24f) | // Disk cl. #2
676         (d22f & d23f) | (d22f & d24f) | (d23f & d24f) |
677         (d31f & d32f) | (d31f & d33f) | (d31f & d34f) | // Disk cl. #3
678         (d32f & d33f) | (d32f & d34f) | (d33f & d34f) |
679         (d41f & d42f) | (d41f & d43f) | (d41f & d44f) | // Disk cl. #4
680         (d42f & d43f) | (d42f & d44f) | (d43f & d44f) |
681         (d51f & d52f) | (d51f & d53f) | (d51f & d54f) | // Disk cl. #5
682         (d52f & d53f) | (d52f & d54f) | (d53f & d54f) |
683         (d61f & d62f) | (d61f & d63f) | (d61f & d64f) | // Disk cl. #6
684         (d62f & d63f) | (d62f & d64f) | (d63f & d64f) |
685         (c11f & c12f) | // Controllers type 1
686         (c21f & c22f) | // Controllers type 2
687         (p11f & p12f) | // Processors type 1
688         (p21f & p22f) ) // Processors type 2
689 endproperties

```

A.11 Oil pipeline or C(k,n: F) system

Summarised version of an IOSA model for the non-Markovian $C(k, n: F)$ repairable system, for $n = 20$ and $k = 3$. These models were used to produce the results presented in Section 4.6.6. The number of system components increases with n , but not with k .

```

1 // These distributions are used in Section 4.1 of José Villén-
2 // Altamirano: RESTART simulation of non-Markov consecutive-k-
3 // out-of-n:F repairable systems, Reliability Engineering and
4 // System Safety, Vol. 95, Issue 3, 2010, pp. 247-254:
5 // - Repair time ~ Lognormal(1.21,0.8)
6 // - Nodes lifetime ~ Exponential(lambda) or Rayleigh(sigma)
7 //
8 //         for (lambda,sigma) in {(0.001 , 798.000),
9 //                                 (0.0003, 2659.615),
10 //                                 (0.0001, 7978.845)}
11 module BE_pipe1
12     c_fail1: clock;
13     c_repair1: clock;
14     inform1: [0..2] init 0; // 0 idle, 1 inform fail, 2 inform repair
15     broken1: [0..2] init 0; // 0 operational, 1 broken, 2 under repair
16     // failing (by itself)

```

```

17     [fpipe1!] broken1==0 & inform1==0 @ c_fail1 -> (inform1'= 1) &
18                                           (broken1'= 1);
19     [fail1!!] inform1 == 1 -> (inform1'= 0);
20     // reparation (with repairman)
21     [repair1??] broken1==1 & inform1==0
22         -> (broken1'= 2) &
23             (c_repair1'= lognormal(1.21,0.8));
24     [rpipe1!] broken1 == 2 @ c_repair1 -> (inform1'= 2) &
25                                           (broken1'= 0) &
26                                           (c_fail1'= rayleigh(729));

27     [repaired1!!] inform1 == 2 -> (inform1'= 0);
28 endmodule

:

370 module BE_pipe20
371     c_fail20: clock;
372     c_repair20: clock;
373     inform20: [0..2] init 0; // 0 idle, 1 inform fail, 2 inform repair
374     broken20: [0..2] init 0; // 0 operational, 1 broken, 2 under repair
375     // failing (by itself)
376     [fpipe20!] broken20==0 & inform20==0 @ c_fail20 -> (inform20'= 1) &
377                                                         (broken20'= 1);
378     [fail20!!] inform20 == 1 -> (inform20'= 0);
379     // reparation (with repairman)
380     [repair20??] broken20==1 & inform20==0
381         -> (broken20'= 2) &
382             (c_repair20'= lognormal(1.21,0.8));
383     [rpipe20!] broken20 == 2 @ c_repair20 -> (inform20'= 2) &
384                                                         (broken20'= 0) &
385                                                         (c_fail20'= rayleigh(729));
386     [repaired20!!] inform20 == 2 -> (inform20'= 0);
387 endmodule

388
389 module Repairman
390     xs[20] : bool init false; // Array of Booleans
391     busy   : bool init false;
392     // Register a failure
393     [ fail1?? ] -> (xs[0]'= true);
394
395     :
396
411     [ fail20?? ] -> (xs[19]'= true);
412     // Begin a repair
413     [ repair1!! ] busy == false & fsteq(xs,true) == 0
414         -> (busy'= true);
415
416     :
417
450     [ repair20!! ] busy == false & fsteq(xs,true) == 19
451         -> (busy'= true);
452     // Finish a repair
453     [ repaired1?? ] -> (busy'= false) & (xs[0]'= false);

```

```
        :  
471     [ repaired20?? ] -> (busy'= false) & (xs[19]'= false);  
472 endmodule  
473  
474 properties  
475     S( ( broken1>0 & broken2>0 & broken3>0 ) |  
476       ( broken2>0 & broken3>0 & broken4>0 ) |  
        :  
491     ( broken17>0 & broken18>0 & broken19>0 ) |  
492     ( broken18>0 & broken19>0 & broken20>0 ) )  
493 endproperties
```

Appendix: Measure theory

B

Some fundamentals of measure theory are epitomised here. These concepts are useful to understand the definitions and results presented in Appendix C, and also to comprehend the more theoretical aspects of Sections 2.3.3, 3.3.4 and 4.4.

All results in this appendix are presented without proof. The interested reader can find a full introduction to measure theory in classical works like [Bre68] and the more modern [Dur10]. Also, N. Vaillant's online tutorial at www.probability.net is highly recommended.

In what follows Ω will denote an arbitrary set and 2^Ω its power set. If $A \in 2^\Omega$ then A^c will denote its complementary set, i.e. $A \cap A^c = \emptyset$ and $A \uplus A^c = \Omega$. The basic building blocks in measure theory are the algebraic structures known as σ -algebra:

Definition 23 (σ -algebra). A σ -algebra over Ω is any collection $\mathcal{F} \subseteq 2^\Omega$ satisfying: $\Omega \in \mathcal{F}$, $A \in \mathcal{F} \Rightarrow A^c \in \mathcal{F}$, and for any denumerable family of subsets of Ω , say $\{\Omega_i\}_{i \in \mathbb{N}}$, its denumerable union is also part of the σ -algebra, viz. $\bigcup_{i \in \mathbb{N}} \Omega_i \in \mathcal{F}$.

If \mathcal{F} is a σ -algebra over Ω , the pair (Ω, \mathcal{F}) is denoted a *measurable space*. The *trivial σ -algebras* of Ω are $\{\emptyset, \Omega\}$ and 2^Ω . The elements of \mathcal{F} in a measurable space (Ω, \mathcal{F}) are called the *measurable sets* of the σ -algebra.

Any collection $\mathcal{C} \subseteq 2^\Omega$ can be turned into a σ -algebra, denote it $\sigma(\mathcal{C})$, by including into $\sigma(\mathcal{C})$ all the complements and denumerable unions of the sets originally in \mathcal{C} . This way one can *generate* a σ -algebra from an arbitrary collection of subsets of Ω . This concept has an alternative definition which we give next.

Definition 24. Let $\mathcal{C} \subseteq 2^\Omega$, then the σ -algebra generated by \mathcal{C} is the intersection of all σ -algebras containing \mathcal{C} , and it is denoted $\sigma(\mathcal{C})$, i.e.

$$\sigma(\mathcal{C}) \doteq \bigcap \left\{ \mathcal{F} \subseteq 2^\Omega \mid \mathcal{C} \subseteq \mathcal{F} \wedge \mathcal{F} \text{ is a } \sigma\text{-algebra} \right\}.$$

Each element of \mathcal{C} is called a *generator*.

Property 11. Let $\mathcal{C} \subseteq 2^\Omega$, then $\sigma(\mathcal{C})$ is a σ -algebra, and in fact it is the minimal σ -algebra containing \mathcal{C} .

Definition 24 provides the means to generate a σ -algebra from arbitrary collections of subsets of Ω . It is also possible to generate a new σ -algebra using σ -algebras as building blocks.

The notion is analogous to the Cartesian product, which for sets $\{\Omega_i\}_{i=1}^n$ and corresponding collections $\{\mathcal{C}_i\}_{i=1}^n$ on their power sets, defines a *rectangle* $A \subseteq \prod_{i=1}^n \Omega_i$ as any set $A = A_1 \times A_2 \times \cdots \times A_n = \prod_{i=1}^n A_i$ s.t. $A_i \in \mathcal{C}_i \cup \{\Omega_i\}$ for all $i = 1, \dots, n$. To obtain a product σ -algebra rather than a product set, it is necessary to work with *measurable rectangles* rather than arbitrary rectangles.

Definition 25. Let $\mathcal{C} = \{(\Omega_i, \mathcal{F}_i)\}_{i=1}^n$ be a finite collection of measurable spaces. A *measurable rectangle* is any rectangle from $\prod_{i=1}^n \Omega_i$ whose constituent sets are measurable sets from $\{\mathcal{F}_i\}_{i=1}^n$.

Definition 26. The *product σ -algebra* of \mathcal{C} from Definition 25, denoted $\bigotimes_{i=1}^n \mathcal{F}_i$, is the σ -algebra generated by the measurable rectangles, viz.

$$\bigotimes_{i=1}^n \mathcal{F}_i \doteq \sigma \left(\left\{ \prod_{i=1}^n A_i \mid A_i \in \mathcal{F}_i \text{ for all } i = 1, 2, \dots, n \right\} \right).$$

Since the product is finite we will also write $\mathcal{F}_1 \otimes \mathcal{F}_2 \otimes \cdots \otimes \mathcal{F}_n = \bigotimes_{i=1}^n \mathcal{F}_i$.

All of the above talks about *structural aspects* of measurability. However, the very name of this theory comes from a more *dynamical* perspective, if you please, involving functions acting over such structures. Measure theory concerns itself with what can be *measured*—and what cannot; at its core lie the concepts of *measure* and *probability measure*.

Definition 27. Let $\mathcal{C} \subseteq 2^\Omega$ s.t. $\emptyset \in \mathcal{C}$ and let $\mu: \mathcal{C} \rightarrow [0, \infty)$. The function μ is a *measure* on \mathcal{C} if $\mu(\emptyset) = 0$ and it is σ -additive, i.e. for any sequence $\{A_i\}_{i \in \mathbb{N}}$ of pairwise disjoint elements of \mathcal{C} where $\biguplus_{i \in \mathbb{N}} A_i \in \mathcal{C}$, μ satisfies $\mu(\biguplus_{i \in \mathbb{N}} A_i) = \sum_{i \in \mathbb{N}} \mu(A_i)$. If besides $\mu(\Omega) = 1$ (and thus $\mu: \mathcal{C} \rightarrow [0, 1]$), then μ is a *probability measure* on \mathcal{C} .

Therefore provided a measurable space (Ω, \mathcal{F}) , a measure μ on \mathcal{F} , and a measurable set $A \in \mathcal{F}$, the measure of A according to μ is $\mu(A) \in [0, \infty)$. The *Dirac probability measure* concentrated on $\{\omega\} \subseteq \Omega$, which we will denote δ_ω , is the unique probability measure s.t. $\delta_\omega(Q) = 1$ if $\omega \in Q$ and $\delta_\omega(Q) = 0$ otherwise, for every $Q \in \mathcal{F}$.

Measurability can be extended to consider the space of functions, bringing forth the concept of *measurable functions* which is defined as follows:

Definition 28. Let $(\Omega_1, \mathcal{F}_1)$ and $(\Omega_2, \mathcal{F}_2)$ be measurable spaces. Then the function $f: \Omega_1 \rightarrow \Omega_2$ is a *measurable function* if the inverse image of every measurable set from \mathcal{F}_2 is a measurable set of \mathcal{F}_1 , i.e.

$$\forall B \in \mathcal{F}_2 . f^{-1}(B) \in \mathcal{F}_1 .$$

The measurability of f in the previous definition is sometimes denoted $f: (\Omega_1, \mathcal{F}_1) \rightarrow (\Omega_2, \mathcal{F}_2)$. Even though Definition 28 might give the impression of being fabricated, several standard and widespread mathematical concepts make use of measurable functions. Probability theory provides a fine example, where a measurable function on a probability space is commonly known as a *random variable*.

This appendix concludes introducing some concepts which will be extensively used along Appendix C. Given a measurable space (Ω, \mathcal{F}) , it is customary to denote by $\Delta(\Omega)$ the set of all probability measures on \mathcal{F} . Furthermore there is a standard construction by [Gir82] to endow $\Delta(\Omega)$ with a σ -algebra. $\Delta(\mathcal{F})$ will thus denote the *Giry σ -algebra* generated by the sets of probability measures

$$\Delta^B(Q) \doteq \{ \mu: \mathcal{F} \rightarrow [0, 1] \mid \mu(Q) \in B \},$$

where $B \in \mathcal{B}([0, 1])$ and $Q \in \mathcal{F}$. Here $\mathcal{B}(\Upsilon)$ is the *Borel σ -algebra* on the set Υ , viz. the σ -algebra generated by the open sets of Υ . The following proposition states that the Giry σ -algebra can be denumerably generated.

Proposition 12. Let (Ω, \mathcal{F}) be a measurable space and denote $\Delta^{\geq q}(Q)$ the set of probability measures $\{ \mu \in \Delta(\Omega) \mid \mu(Q) \geq q \}$ for any $Q \in \mathcal{F}$ and $q \in [0, 1]$. Then

$$\Delta(\mathcal{F}) = \sigma \left(\left\{ \Delta^{\geq q}(Q) \mid q \in \mathbb{Q} \cap [0, 1] \right\} \right).$$

Appendix:

Nondeterministic LMP

C

Nondeterministic Labelled Markov Processes (NLMP, [DSW12]) are the result of several efforts to provide the theory of *labelled Markov processes* from [Des99, DEP02] with internal nondeterminism. NLMP stand out among other approaches seeking the same goal, because they follow the same strategy than Desharnais et al. in [DEP02], who rely on the sound foundations provided by measure theory—see Appendix B.

The general goal is to extend the modelling capabilities of Markov processes with: continuous state spaces; continuous time evolution; external nondeterminism, i.e. governed by the environment; and internal nondeterminism, i.e. decided upon by each process. The formalism of labelled Markov processes covers the first three items, using a labelled set of actions to encode interactions with the environment. This formalism defines reactive models where different transition probabilities are enabled for each action. Thus uncertainty is (only) considered to be probabilistic.

Definition 29 (LMP, [DEP02]). A *labelled Markov process* (LMP) is a tuple $(S, \Sigma, \{\tau_a \mid a \in \mathcal{L}\})$ where (S, Σ) is a measurable space and, for each action label $a \in \mathcal{L}$, the *transition probability function* $\tau_a: S \rightarrow \Delta(S) \cup \{\mathbf{0}\}$ is a measurable function, where $\mathbf{0}: \Sigma \rightarrow [0, 1]$ denotes the *null measure* s.t. $\mathbf{0}(Q) = 0$ for all $Q \in \Sigma$.

The value $\tau_a(s)(Q) \in [0, 1]$ represents the probability of making a transition to any state in Q , provided that the system is in state s and that the action label a has been accepted. Therefore, the transition probability is actually a conditional probability, where the probability of Q is conditioned on the facts that the system is in state s and it actually reacts to action a . Originally [Des99] allow $\tau_a(s)$ to be a *subprobability measure*, i.e. it could happen that $\tau_a(s)(S) < 1$. Instead and following [BDSW14], when action a is refused we let $\tau_a(s) = \mathbf{0}$.

Nondeterministic Labelled Markov Processes were introduced in [DSW12, DWTC09] as a generalisation of LMP to include internal nondeterminism.

Specifically, they allow equally labelled transition probabilities to leave out the same state. Two constraints required by the NLMP formalism set it apart from other approaches which pursue similar goals:

- (a) the transition function maps states to *measurable sets* of probability measures, and
- (b) each transition function must be measurable.

Constraint (a) is motivated by the use of *schedulers* to resolve nondeterminism. Allowing arbitrary target sets of measures could make the theory suffer from measurability issues, namely the decisions to take future actions could be unquantifiable. Constraint (b) is related to the use of modal operators, like the ones LMP allow. Dealing with non-measurable transition functions could render infeasible the measurement of certain execution traces. For examples illustrating these motivations see [Wol12,BDSW14].

Definition 30 (NLMP, [DSW12]). A *nondeterministic labelled Markov process* (NLMP) is a tuple $(S, \Sigma, \{\mathcal{T}_a \mid a \in \mathcal{L}\})$ where:

- (S, Σ) is a measurable space,
- for each label $a \in \mathcal{L}$ the *nondeterministic transition function* $\mathcal{T}_a: S \rightarrow \Delta(\Sigma)$ is measurable.

Notice that the measurability requirement of \mathcal{T}_a requires the definition of a σ -algebra over its codomain, the Giry σ -algebra $\Delta(\Sigma)$. Such definition is a key construction for the development of the NLMP formalism.

Definition 31 (Hit σ -algebra). Let (S, Σ) be as in Definition 30, then $H(\Delta(\Sigma))$ is the minimal σ -algebra containing all sets

$$H_\xi \doteq \{\zeta \in \Delta(\Sigma) \mid \zeta \cap \xi \neq \emptyset\}$$

for the measurable sets $\xi \in \Delta(\Sigma)$.

In Definition 31, H_ξ contains all measurable sets that *hit* the measurable set ξ . Also observe that $\mathcal{T}_a^{-1}(H_\xi)$ is the set of all states $s \in S$ which, through label a , *hit* the set of measures ξ . Thus, resuming the analysis of Definition 30, for each label $a \in \mathcal{L}$ the corresponding nondeterministic transition function \mathcal{T}_a must be measurable from the σ -algebra of states to the hit σ -algebra of measures, i.e. $\mathcal{T}_a: (S, \Sigma) \rightarrow (\Delta(\Sigma), H(\Delta(\Sigma)))$.

As might be expected, LMP are a special case of NLMP where \mathcal{T}_a is the singleton set $\{\tau_a\}$ for every label $a \in \mathcal{L}$. Of course, that requires for single

probability measures to be measurable in the Giry σ -algebra, viz. $\Delta(\Sigma)$ must distinguish points. That condition provided, it can be verified that \mathcal{T}_a is measurable if and only if τ_a is also measurable [BDSW14].

In spite of the structure provided over the state space by Definition 30, the theory can still suffer from measurability issues derived from an *improper* (but anyway *allowed* by the definition) use of the labels. Because of this NLMP have been extended in [Bud12] to provide structure to the label set \mathcal{L} . Thus in addition to the measurable space of states, (S, Σ) , there is a measurable space of labels, (\mathcal{L}, Λ) . The resulting *transition function* \mathcal{T} then maps states to measurable sets of the product σ -algebra $\Lambda \otimes \Delta(\Sigma)$, and the hit σ -algebra on which the measurability of \mathcal{T} depends is defined by means of measurable rectangles.

Much of the theory of Nondeterministic Labelled Markov Processes is devoted to the development of bisimulation relations with different degrees of observability. Bisimilarity as defined by [Mil80] for LTS can be extended over the much more complex world of LMP in more than one way. Two notions have been developed by Desharnais et al., the first of which is defined directly on states [Des99]. Therefore this definition can separate systems which could be potentially indistinguishable from the point of view of Σ . Later in [DDL06] an “event-wise” bisimulation relation is introduced, hence providing a notion of behavioural equivalence more attune to the measure-theoretic definition of LMP. Making reference to the way in which these relations are defined, [DDL06] denotes the former (point-wise) relations *state bisimulations*, and the latter (event-wise) relations *event bisimulations*.

NLMP have their own analogous state and event bisimulation relations. There is also a third notion, denoted *hit bisimulation* in [BDSW14], whose coarseness of observability lies somewhere in between the other two. Interestingly, all these notions coincide under special conditions. However and in general the state bisimilarity is (strictly) the finest and the event bisimilarity is (strictly) the coarsest [BDSW14]. All these notions apply also to NLMP with structure over the label set [Bud12].

Bibliography

- [Bar84] H. P. Barendregt. *The Lambda Calculus: Its Syntax and Semantics*. Sole Distributors for the U.S.A. And Canada, Elsevier Science Pub. Co., 1984.
- [Bar14] Benoît Barbot. *Acceleration for statistical model checking*. PhD thesis, École normale supérieure de Cachan, France, 2014.
- [Bay70] A. J. Bayes. Statistical techniques for simulation models. *Australian Computer Journal*, 2(4):180–184, 1970.
- [Bay72] A. J. Bayes. A minimum variance sampling technique for simulation models. *J. ACM*, 19(4):734–741, 1972.
- [BCC⁺14] Tomáš Brázdil, Krishnendu Chatterjee, Martin Chmelik, Vojtech Forejt, Jan Kretínský, Marta Z. Kwiatkowska, David Parker, and Mateusz Ujma. Verification of markov decision processes using learning algorithms. In *ATVA 2014*, volume 8837 of *LNCS*, pages 98–114. Springer, 2014.
- [BDH15] Carlos E. Budde, Pedro R. D’Argenio, and Holger Hermanns. Rare event simulation with fully automated importance splitting. In *EPEW 2015*, volume 9272 of *LNCS*, pages 275–290. Springer, 2015.
- [BDH⁺17] Carlos E. Budde, Christian Dehnert, Ernst Moritz Hahn, Arnd Hartmanns, Sebastian Junges, and Andrea Turrini. JANI: quantitative model and tool interaction. In *TACAS 2017*, volume 10206 of *LNCS*, pages 151–168, 2017.
- [BDL⁺06] Gerd Behrmann, Alexandre David, Kim Guldstrand Larsen, John Håkanesson, Paul Pettersson, Wang Yi, and Martijn Hendriks. UPPAAL 4.0. In *QEST 2006*, pages 125–126. IEEE Computer Society, 2006.
- [BDM17] Carlos E. Budde, Pedro R. D’Argenio, and Raúl E. Monti. Compositional construction of importance functions in fully auto-

- mated importance splitting. In *VALUETOOLS 2016*. ACM, ACM, 2017.
- [BDSW14] Carlos E. Budde, Pedro R. D’Argenio, Pedro Sánchez Terraf, and Nicolás Wolovick. A theory for the semantics of stochastic and non-deterministic continuous systems. In *ROCKS 2012*, volume 8453 of *LNCS*, pages 67–86. Springer, 2014.
- [Bel57] Richard Bellman. A markovian decision process. Technical report, DTIC Document, 1957.
- [BFG⁺97] R.I. Bahar, E.A. Frohm, C.M. Gaona, G.D. Hachtel, E. Macii, A. Pardo, and F. Somenzi. Algebraic decision diagrams and their applications. *Formal Methods in System Design*, 10(2):171–206, 1997.
- [BFHH11] Jonathan Bogdoll, Luis María Ferrer Fioriti, Arnd Hartmanns, and Holger Hermanns. Partial order methods for statistical model checking and simulation. In *FMOODS 2011 & FORTE 2011*, volume 6722 of *LNCS*, pages 59–74. Springer, 2011.
- [BGC04] Christel Baier, Marcus Größer, and Frank Ciesinski. Partial order reduction for probabilistic systems. In *QEST 2004* [DBL04], pages 230–239.
- [Bil12] P. Billingsley. *Probability and Measure*. Wiley Series in Probability and Statistics. Wiley, 2012.
- [BK08] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. MIT press Cambridge, 2008.
- [BKH99] Christel Baier, Joost-Pieter Katoen, and Holger Hermanns. Approximate symbolic model checking of continuous-time markov chains. In *CONCUR 1999*, volume 1664 of *LNCS*, pages 146–161. Springer, 1999.
- [Bre68] L. Breiman. *Probability*. Classics in Applied Mathematics. Society for Industrial and Applied Mathematics, 1968.
- [Bry86] Randal E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Trans. Comput.*, 35(8):677–691, August 1986.

- [Bud12] Carlos E. Budde. No-determinismo completamente medible en procesos probabilísticos continuos. Master's thesis, Universidad Nacional de Córdoba, Argentina, 2012.
- [BvdP02] Stefan Blom and Jaco van de Pol. State space reduction by proving confluence. In *CAV 2002* [DBL02], pages 596–609.
- [CAB05] J. Ching, S.K. Au, and J.L. Beck. Reliability estimation for dynamical systems subject to stochastic excitation using subset simulation with splitting. *Computer Methods in Applied Mechanics and Engineering*, 194(12–16):1557 – 1579, 2005. Special Issue on Computational Methods in Stochastic Mechanics and Reliability Analysis.
- [CDMFG12] F. Cérou, P. Del Moral, T. Furon, and A. Guyader. Sequential Monte Carlo for rare event estimation. *Statistics and Computing*, 22(3):795–808, 2012.
- [CE81] Edmund M. Clarke and E. Allen Emerson. *Design and Synthesis of Synchronization Skeletons Using Branching-Time Temporal Logic*, volume 131 of *LNCS*, pages 52–71. Springer, 1981.
- [CES86] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. Program. Lang. Syst.*, 8(2):244–263, April 1986.
- [CFM⁺93] Edmund M. Clarke, M. Fujita, P. C. McGeer, K. McMillan, JC-Y. Yang, and X Zhao. Multi-terminal binary decision diagrams: An efficient data structure for matrix representation. 1993.
- [CG07] Frédéric Cérou and Arnaud Guyader. Adaptive multilevel splitting for rare event analysis. *Stochastic Analysis and Applications*, 25(2):417–443, 2007.
- [CR65] Y. S. Chow and Herbert Robbins. On the asymptotic theory of fixed-width sequential confidence intervals for the mean. *The Annals of Mathematical Statistics*, 36(2):457–462, 04 1965.
- [CW96] Edmund M. Clarke and Jeannette M. Wing. Formal methods: State of the art and future directions. *ACM Comput. Surv.*, 28(4):626–643, 1996.

- [dAKN⁺00] Luca de Alfaro, Marta Z. Kwiatkowska, Gethin Norman, David Parker, and Roberto Segala. Symbolic model checking of probabilistic processes using mtbdds and the kronecker representation. In *TACAS 2000*, volume 1785 of *LNCS*, pages 395–410. Springer, 2000.
- [DBL02] *CAV 2002*, volume 2404 of *LNCS*. Springer, 2002.
- [DBL04] *QEST 2004*. IEEE Computer Society, 2004.
- [DD09] Thomas Dean and Paul Dupuis. Splitting for rare event simulation: A large deviation approach to design and analysis. *Stochastic Processes and their Applications*, 119(2):562 – 587, 2009.
- [DDL06] Vincent Danos, Josee Desharnais, François Laviolette, and Prakash Panangaden. Bisimulation and cocongruence for probabilistic systems. *Inf. Comput.*, 204(4):503–523, 2006.
- [DEP02] Josee Desharnais, Abbas Edalat, and Prakash Panangaden. Bisimulation for labelled Markov processes. *Inf. Comput.*, 179(2):163–193, 2002.
- [Des99] Josée Desharnais. *Labelled markov processes*. PhD thesis, McGill University, Montréal, 1999.
- [DHLS16] Pedro R. D’Argenio, Arnd Hartmanns, Axel Legay, and Sean Sedwards. Statistical approximation of optimal schedulers for probabilistic timed automata. In *IFM 2016*, volume 9681 of *LNCS*, pages 99–114. Springer, 2016.
- [DJJL02] Pedro R. D’Argenio, Bertrand Jeannet, Henrik Ejersbo Jensen, and Kim Guldstrand Larsen. Reduction and refinement strategies for probabilistic analysis. volume 2399 of *LNCS*, pages 57–76. Springer, 2002.
- [DJKV16] Christian Dehnert, Sebastian Junges, Joost-Pieter Katoen, and Matthias Volk. The probabilistic model checker storm (extended abstract). *CoRR*, abs/1610.08713, 2016.
- [DK05] Pedro R. D’Argenio and Joost-Pieter Katoen. A theory of stochastic systems part I: Stochastic automata. *Inf. Comput.*, 203(1):1–38, 2005.

- [DLM16] Pedro R. D’Argenio, Matías David Lee, and Raúl E. Monti. Input/Output Stochastic Automata - Compositionality and Determinism. In *FORMATS 2016*, volume 9884 of *LNCS*, pages 53–68. Springer, Springer, 2016.
- [DLST15] Pedro D’Argenio, Axel Legay, Sean Sedwards, and Louis-Marie Traonouez. Smart sampling for lightweight verification of markov decision processes. *STTT*, 17(4):469–484, 2015.
- [DN04] Pedro R. D’Argenio and Peter Niebert. Partial order reduction on concurrent probabilistic programs. In *QEST 2004* [DBL04], pages 240–249.
- [DSW12] Pedro R. D’Argenio, Pedro Sánchez Terraf, and Nicolás Wolovick. Bisimulations for non-deterministic labelled Markov processes. *Mathematical Structures in Computer Science*, 22(1):43–68, 2012.
- [dt04] The Coq development team. *The Coq proof assistant reference manual*. LogiCal Project, 2004. Version 8.0.
- [Dur10] R. Durrett. *Probability: Theory and Examples*. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press, 2010.
- [dVRLR09] Miguel de Vega Rodrigo, Guy Latouche, and Marie-Ange Remiche. Modeling bufferless packet-switching networks with packet dependencies. *Computer Networks*, 53(9):1450–1466, 2009.
- [DWTC09] Pedro R. D’Argenio, Nicolás Wolovick, Pedro Sánchez Terraf, and Pablo Celayes. Nondeterministic labeled markov processes: Bisimulations and logical characterization. In *QEST 2009*, pages 11–20. IEEE Computer Society, 2009.
- [FHH⁺11] Martin Fränzle, Ernst Moritz Hahn, Holger Hermanns, Nicolás Wolovick, and Lijun Zhang. Measurability and safety verification for stochastic hybrid systems. In *HSCC 2011*, pages 43–52. ACM, 2011.
- [Gar00] Marnix Joseph Johann Garvels. *The splitting method in rare event simulation*. PhD thesis, Department of Computer Science, University of Twente, 2000.

- [GHML11] Arnaud Guyader, Nicolas Hengartner, and Eric Matzner-Løber. Simulation and estimation of extreme quantiles and extreme probabilities. *Applied Mathematics & Optimization*, 64(2):171–196, 2011.
- [GHSZ98] Paul Glasserman, Philip Heidelberger, Perwez Shahabuddin, and Tim Zajic. A large deviations perspective on the efficiency of multilevel splitting. *IEEE Transactions on Automatic Control*, 43(12):1666–1679, 1998.
- [GHSZ99] Paul Glasserman, Philip Heidelberger, Perwez Shahabuddin, and Tim Zajic. Multilevel splitting for estimating rare event probabilities. *Operations Research*, 47(4):585–600, 1999.
- [GI89] Peter W. Glynn and Donald L. Iglehart. Importance sampling for stochastic simulations. *Management Science*, 35(11):1367–1392, 1989.
- [Gir82] Michèle Giry. *A categorical approach to probability theory*, pages 68–85. Springer Berlin Heidelberg, Berlin, Heidelberg, 1982.
- [GK98] Marnix J. J. Garvels and Dirk P. Kroese. A comparison of RESTART implementations. In *WSC 1998*, pages 601–608. WSC, 1998.
- [GRT09] Peter W. Glynn, Gerardo Rubino, and Bruno Tuffin. *Robustness Properties and Confidence Interval Reliability Issues*, pages 63–84. In Rubino and Tuffin [RT09b], 2009.
- [GSH⁺92] A. Goyal, P. Shahabuddin, P. Heidelberger, V. F. Nicola, and P. W. Glynn. A unified framework for simulating markovian models of highly dependable systems. *IEEE Transactions on Computers*, 41(1):36–51, Jan 1992.
- [GVOK02] Marnix J. J. Garvels, Jan-Kees C. W. Van Ommeren, and Dirk P. Kroese. On the importance function in splitting simulation. *European Transactions on Telecommunications*, 13(4):363–371, 2002.
- [Har15] Arnd Hartmanns. *On the analysis of stochastic timed systems*. PhD thesis, Universität des Saarlandes, Postfach 151141, 66041 Saarbrücken, 2015.

- [Hei95] Philip Heidelberger. Fast simulation of rare events in queueing and reliability models. *ACM Trans. Model. Comput. Simul.*, 5(1):43–85, 1995.
- [HHHK13] Ernst Moritz Hahn, Arnd Hartmanns, Holger Hermanns, and Joost-Pieter Katoen. A compositional modelling and analysis framework for stochastic hybrid systems. *Formal Methods in System Design*, 43(2):191–232, 2013.
- [HJ94] Hans Hansson and Bengt Jonsson. A logic for reasoning about time and reliability. *Formal Aspects of Computing*, 6(5):512–535, 1994.
- [HMZ⁺12] David Henriques, João Martins, Paolo Zuliani, André Platzer, and Edmund M. Clarke. Statistical model checking for markov decision processes. In *QEST 2012*, pages 84–93. IEEE Computer Society, 2012.
- [JLS13] Cyrille Jégourel, Axel Legay, and Sean Sedwards. Importance splitting for statistical model checking rare properties. In *CAV 2013*, volume 8044 of *LNCS*, pages 576–591. Springer, 2013.
- [JLST15] Cyrille Jégourel, Axel Legay, Sean Sedwards, and Louis-Marie Traonouez. Distributed verification of rare properties with lightweight importance splitting observers. *CoRR*, abs/1502.01838, 2015.
- [JS06] Sandeep Juneja and Perwez Shahabuddin. Rare-event simulation techniques: An introduction and recent advances. *Handbooks in Operations Research and Management Science*, 13:291–350, 2006.
- [KH51] Herman Kahn and Ted E. Harris. Estimation of particle transmission by random sampling. *National Bureau of Standards applied mathematics series*, 12:27–30, 1951.
- [KN99] Dirk P Kroese and Victor F Nicola. Efficient estimation of overflow probabilities in queues with breakdowns. *Performance Evaluation*, 36:471–484, 1999.
- [KNP07] Marta Z. Kwiatkowska, Gethin Norman, and David Parker. Stochastic model checking. In *SFM 2007*, volume 4486 of *LNCS*, pages 220–270. Springer, 2007.

- [KNP11] Marta Z. Kwiatkowska, Gethin Norman, and David Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *CAV 2011*, volume 6806 of *LNCS*, pages 585–591. Springer, 2011.
- [Kon80] J. M. Kontoleon. Reliability determination of a r-successive-out-of-n:f system. *IEEE Transactions on Reliability*, R-29(5):437–437, 1980.
- [LDB10] Axel Legay, Benoît Delahaye, and Saddek Bensalem. Statistical model checking: An overview. In *RV 2010*, volume 6418 of *LNCS*, pages 122–135. Springer, 2010.
- [LDT07] Pierre L’Ecuyer, Valérie Demers, and Bruno Tuffin. Rare events, splitting, and quasi-Monte Carlo. *ACM Trans. Model. Comput. Simul.*, 17(2), April 2007.
- [LK00] A.M. Law and W.D. Kelton. *Simulation modeling and analysis*. McGraw-Hill series in industrial engineering and management science. McGraw-Hill, 2000.
- [LLGLT09] Pierre L’Ecuyer, François Le Gland, Pascal Lezaud, and Bruno Tuffin. *Splitting Techniques*, pages 39–61. In Rubino and Tuffin [RT09b], 2009.
- [LMT09] Pierre L’Ecuyer, Michel Mandjes, and Bruno Tuffin. *Importance Sampling in Rare Event Simulation*, pages 17–38. In Rubino and Tuffin [RT09b], 2009.
- [LST14] Axel Legay, Sean Sedwards, and Louis-Marie Traonouez. Scalable verification of markov decision processes. In *SEFM 2014*, volume 8938 of *LNCS*, pages 350–362. Springer, 2014.
- [LT11] Pierre L’Ecuyer and Bruno Tuffin. Approximating zero-variance importance sampling in a reliability setting. *Annals of Operations Research*, 189(1):277–297, 2011.
- [LZ00] Yeh Lam and Yuan Lin Zhang. Repairable consecutive-k-out-of-n: F system with markov dependence. *Naval Research Logistics (NRL)*, 47(1):18–39, 2000.
- [Mil80] Robin Milner. *A Calculus of Communicating Systems*, volume 92 of *LNCS*. Springer, 1980.

- [Mil89] Robin Milner. *Communication and concurrency*. PHI Series in computer science. Prentice Hall, 1989.
- [Nor98] J.R. Norris. *Markov Chains*. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press, 1998.
- [Oxf13] Oxford Dictionaries. *Oxford English Dictionary*. Oxford University Press, 7 edition, 2013.
- [Par02] David Anthony Parker. *Implementation of symbolic model checking for probabilistic systems*. PhD thesis, University of Birmingham, 2002.
- [Pnu77] Amir Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977*, pages 46–57. IEEE Computer Society, 1977.
- [PTVF07] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes*. Cambridge University Press, 3 edition, 2007.
- [RdBS16] Daniël Reijbergen, Pieter-Tjerk de Boer, and Werner R. W. Scheinhardt. Hypothesis testing for rare-event simulation: Limitations and possibilities. In *ISoLA 2016*, volume 9952 of *LNCS*, pages 16–26, 2016.
- [RdBSh13] Daniël Reijbergen, Pieter-Tjerk de Boer, Werner Scheinhardt, and Boudewijn Haverkort. Automated Rare Event Simulation for Stochastic Petri Nets. In *QEST 2013*, volume 8054 of *LNCS*, pages 372–388. Springer, 2013.
- [RDDA15] Germán Regis, Renzo Degiovanni, Nicolás D’Ippolito, and Nazareno Aguirre. Specifying event-based systems with a counting fluent temporal logic. In *ICSE 2015*, pages 733–743. IEEE Computer Society, 2015.
- [Ric06] J.A. Rice. *Mathematical Statistics and Data Analysis*. Cengage Learning, 2006.
- [RS15] Enno Ruijters and Mariëlle Stoelinga. Fault tree analysis: A survey of the state-of-the-art in modeling, analysis and tools. *Computer Science Review*, 15:29–62, 2015.

- [RT09a] Gerardo Rubino and Bruno Tuffin. *Introduction to Rare Event Simulation*, pages 1–13. In [RT09b], 2009.
- [RT09b] Gerardo Rubino and Bruno Tuffin, editors. *Rare Event Simulation Using Monte Carlo Methods*. John Wiley & Sons, Ltd, 2009.
- [SS07] Murray R. Spiegel and Larry J. Stephens. *Schaum's Outline of Statistics*. McGraw-Hill, 2007.
- [Tso92] Pantelis Tsoucas. Rare events in series of queues. *Journal of Applied Probability*, 29(1):168–175, July 1992.
- [VA98] José Villén-Altamirano. RESTART method for the case where rare events can occur in retrials from any threshold. *International Journal of Electronics and Communications*, 52:183–189, 1998.
- [VA07a] José Villén-Altamirano. Importance functions for RESTART simulation of highly-dependable systems. *Simulation*, 83(12):821–828, 2007.
- [VA07b] José Villén-Altamirano. Rare event RESTART simulation of two-stage networks. *European J. of Operational Research*, 179(1):148–159, 2007.
- [VA09] José Villén-Altamirano. RESTART simulation of networks of queues with erlang service times. In *WSC 2009*, pages 1146–1154. WSC, 2009.
- [VA10] José Villén-Altamirano. RESTART simulation of non-markov consecutive-k-out-of-n: F repairable systems. *Rel. Eng. & Sys. Safety*, 95(3):247–254, 2010.
- [VA14] José Villén-Altamirano. Asymptotic optimality of RESTART estimators in highly dependable systems. *Reliability Engineering & System Safety*, 130:115 – 124, 2014.
- [Val90] Antti Valmari. A stubborn attack on state explosion. In *CAV 1990*, volume 531 of *LNCS*, pages 156–165. Springer, 1990.
- [VAMGF94] Manuel Villén-Altamirano, A. Martínez-Marrón, J. Gamo, and F. Fernández-Cuesta. Enhancement of the accelerated simulation method RESTART by considering multiple thresholds. In *Proc. 14th Int. Teletraffic Congress*, pages 797–810, 1994.

- [VAVA91] Manuel Villén-Altamirano and José Villén-Altamirano. RE-START: a method for accelerating rare event simulations. In *Queueing, Performance and Control in ATM (ITC-13)*, pages 71–76. Elsevier, 1991.
- [VAVA02] Manuel Villén-Altamirano and José Villén-Altamirano. Analysis of restart simulation: Theoretical basis and sensitivity study. *European Transactions on Telecommunications*, 13(4):373–385, 2002.
- [VAVA06] Manuel Villén-Altamirano and José Villén-Altamirano. On the efficiency of restart for multidimensional state systems. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 16(3):251–279, 2006.
- [VAVA11] Manuel Villén-Altamirano and José Villén-Altamirano. The rare event simulation method RESTART: efficiency analysis and guidelines for its application. In *Network Performance Engineering*, volume 5233 of *LNCS*, pages 509–547. Springer, 2011.
- [VAVA13] Manuel Villén-Altamirano and José Villén-Altamirano. Rare event simulation of non-markovian queueing networks using RE-START method. *Simulation Modelling Practice and Theory*, 37:70 – 78, 2013.
- [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.
- [Wei84] Mark Weiser. Program slicing. *IEEE Trans. Software Eng.*, 10(4):352–357, 1984.
- [Wil27] Edwin B. Wilson. Probable inference, the law of succession, and statistical inference. *Journal of the American Statistical Association*, 22(158):209–212, 1927.
- [Wol12] Nicolás Wolovick. *Continuous Probability and Nondeterminism in Labeled Transition Systems*. PhD thesis, Universidad Nacional de Córdoba, Argentina, 2012.

- [XLL07] Gang Xiao, Zhizhong Li, and Ting Li. Dependability estimation for non-markov consecutive-k-out-of-n: F repairable systems by fast simulation. *Reliability Engineering & System Safety*, 92(3):293 – 299, 2007. Selected Papers Presented at the Fourth International Conference on Quality and Reliability, ICQR 2005, Fourth International Conference on Quality and Reliability.
- [Yi90] Wang Yi. Real-time behaviour of asynchronous agents. In *CONCUR '90*, volume 458 of *LNCS*, pages 502–520. Springer, 1990.
- [YS02] Håkan L. S. Younes and Reid G. Simmons. Probabilistic verification of discrete event systems using acceptance sampling. In *CAV 2002* [DBL02], pages 223–235.
- [ZM12] Armin Zimmermann and Paulo Maciel. Importance function derivation for restart simulations of petri nets. In *RESIM 2012*, pages 8–15, Trondheim, Norway, 2012.
- [ZRWCL16] Armin Zimmermann, Daniël Reijbergen, Alexander Wichmann, and Andrés Canabal Lavista. Numerical results for the automated rare event simulation of stochastic Petri nets. In *RESIM 2016*, pages 1–10, Eindhoven, Netherlands, 2016.