UNIVERSITE BRETAGNE LOIRE / MATHSTIC

UNIVERSITÉ DE NANTES

HABILITATION À DIRIGER DES RECHERCHES DE

L'UNIVERSITÉ DE NANTES

COMUE UNIVERSITÉ BRETAGNE LOIRE

ÉCOLE DOCTORALE N° 601
Mathématiques et Sciences et Technologies de l'Information et de la Communication
Spécialité : Informatique

Par

Benoît DELAHAYE

Modeling and Verification of Systems with Uncertainties

Habilitation présentée et soutenue à Nantes, le 10 décembre 2020
Unité de recherche : LS2N CNRS UMR 6004

Rapporteurs avant soutenance :

Béatrice Bérard	Professeur émérite, Université Pierre et Marie Curie
Pedro R. D'Argenio	Professeur, Nacional Universidad de Córdoba
Holger Hermanns	Professeur, Saarland University

Composition du Jury :

Attention, en cas d'absence d'un des membres du Jury le jour de la soutenance, la composition du jury doit être revue pour s'assurer qu'elle est conforme et devra être répercutée sur la couverture de thèse

Président :	Colin De La Higuera	Professeur, Université de Nantes
Examinateurs :	Christel Baier	Professeur, Technische Universität Dresden
	Béatrice Bérard	Professeur émérite, Université Pierre et Marie Curie
	Patricia Bouyer-Decitre	Directrice de Recherche, CNRS, Ecole Normale Paris Saclay
	Pedro R. D'Argenio	Professeur, Nacional Universidad de Córdoba
	Holger Hermanns	Professeur, Saarland University

"I, at any rate, am convinced that [God] does not throw dice".

Albert Einstein

ACKNOWLEDGEMENT

First, I would like to thank my reviewers and Jury. I am truly honoured that scientists whom I admire and respect, both for their scientific contributions and for their personality, have accepted to review my work and listen to my presentation. I could not have dreamt of a better jury.

I would also like to thank all my co-authors, without whom this document would be much shorter. In particular, I would like to thank my former Ph.D. students and postdocs: Amine, Eva, Dimitri, Hadrien and Paulin. It has been an honour working with you, and although none of you have (yet) pursued in Academia, it would be my pleasure to work again at your side. Thank you Anicet for the great work we did together. Although you were not officially my Ph.D. student, I would have been proud to consider you as such. There are of course too many names to cite here, but I will insist on a few names: Thank you Etienne for making me part of the ANR PACS crew, and also for the time spent on PIPTAS (what a name...). Thank you Didier for always being available for scientific (or non-scientific) discussions and also for the prank you played at QEST'17 – I will remember. Thank you Kim for taking me as a postdoc and spending a lot of your precious time working with me.

Thank you to all my colleagues at LS2N and beyond (who also might be co-authors, but I hope they can forgive me for thanking them twice). I spend more than half of my days at work and I could not do so without you. In particular, thank you to the AeLoS team for taking me in, thank you to Christian for being the best team leader ever, thank you to Arnaud and Charlotte for being the best officemates ever, thank you to Pascal for so many great discussions (and also for proof-reading this manuscript), and of course thank you to Claude for pushing me to apply in Nantes and supporting me since: best idea ever. Thank you Colin for making me welcome in Nantes (I will remember the first words you ever said to me: "Toi, c'est les automates probabilistes"). Thank you to the ComBi team for giving me so many occasions to drink beers: Audrey, Géraldine, Guillaume (who also proof-read this manuscript), Jérémie, Samuel. A special thank you to Damien E. for so many trips, beers, discussions, coffee-breaks, and often all of them at once. Thank you for your enthusiasm and for convincing me that model-checking and verification can be interesting to others than computer-scientists. I know we don't agree on the nature of probabilities, but it will be my pleasure to try to convince you again over our next beer.

Merci à mes collègues du département informatique, en particulier à ceux qui se sacrifient pour prendre les responsabilités dont personne ne veut : Emmanuel, Eric, Laura, Salima...
Je tiens aussi à remercier les nombreux et nombreuses collègues qui font en sorte que nous puissions faire notre travail dans les meilleures conditions, qui nous simplifient la vie au maximum et réparent nos bêtises : Anne-Françoise, Annie, Elodie, Caroline, Charlery, Christine, Damien V., Jean-Yves, Karine, Marion, Sophie, Virginie, Vivian et j'en oublie probablement...

Finalement, un grand merci à mes amis et à ma famille, qui me supportent et me soutiennent en permanence depuis toujours.

TABLE OF CONTENTS

LIST OF TABLES AND FIGURES

INTRODUCTION

Science[1]

Science is a systematic enterprise that builds and organizes knowledge in the form of testable explanations and predictions about the universe. [...] **Modern science** is typically divided into three major branches that consist of the *natural sciences* (*e.g.*, biology, chemistry, and physics), which study nature in the broadest sense; the social sciences (*e.g.*, economics, psychology, and sociology), which study individuals and societies; and the formal sciences (*e.g.*, logic, mathematics, and theoretical computer science), which study abstract concepts. [...]

As one can see from the above definition, the aim of Science, in general, is to gather and organize knowledge about given objects of study (which we will call *systems* in the rest of the document). As varied as these systems can be, a common concept is used across all modern science branches in order to represent our understanding of these systems: the concept of *model*. A model is in fact the abstract representation that is carefully built from the gathered knowledge on the system and then used instead of the system itself in order to perform analyses or prediction. When a scientist studies a system through its model, his/her goal is to draw conclusions/predictions about the system itself that are as little biased as possible by the abstraction performed to obtain the model from the "real" system or the perception of it. Although a model is by definition necessarily imperfect, as it is an abstraction that only reflects our understanding (or point of view) of the system, the aim of scientists is to make it as accurate (*i.e.*, faithful to the system) as possible while preserving our ability to use it in our studies. Unfortunately, these two goals are usually antinomic: the more precise the model is, the more difficult it is to use for prediction and analysis for computational reasons.

While natural and social sciences mostly use models as a tool for representing and analyzing given systems, some branches of formal sciences consider the models themselves as their objects of study. For instance, some branches of theoretical computer science are dedicated to developing

1. Taken from `https://en.wikipedia.org/wiki/Science`

languages, formalisms, theories, techniques and tools to build and analyze models, with *a priori* no restriction on the systems these models represent. This is the broad context in which the contributions presented in this document lie.

Modeling with uncertainties

As explained above, the point of modeling a given system is to represent the knowledge we have of this system and use the model for analysis or prediction. Most of the time, the knowledge we have is incomplete, which is the reason why analysis is required in the first place. There are then two ways to account for this incomplete knowledge: (1) *Deterministic modeling* and (2) *Modeling with uncertainties*. Deterministic modeling amounts to only including in the models what is known about the systems, while abstracting away all that is unknown. In this case, the model is an imperfect abstraction of the system and the modeler knows that the results obtained through its analysis might be imperfect themselves, without specific information about these imperfections. On the other hand, modeling with uncertainties amounts to including the uncertainties we have about the system inside of the model. In this case, the model is an abstraction of the imperfect system perception and its analysis yields information not only about the system itself but also about the approximations that have been performed during abstraction in order to obtain the model.

Example 1. As an illustration, assume that one wants to study the time that a train takes to go from Paris to Nantes. If we deliberately oversimplify the problem, we know that this duration can be computed as the ratio between the distance from Paris to Nantes and the speed of the train. While the distance from Paris to Nantes is perfectly known (342 km), we do not know the exact speed of the train. Indeed, depending on the type of train and the weather conditions, this speed can vary.
Deterministic modeling, in this case, would amount to measuring the average speed of trains going from Paris to Nantes (150 km/h) and then using this value in the model. In this case, the model would allow us to conclude that the time taken by a train from Paris to Nantes is approximately 2h16.
On the other hand, **modeling with uncertainties** would require to take into account in the model that the speed is not perfectly known. This could be done in several ways (the following description is not exhaustive):

- The first way could be to include in the model a **choice** between high-speed and low-speed

trains, thus leading to a *non-deterministic model* that will yield different outputs depending on the choice of the train.

- Another way of taking this uncertainty into account could be to measure the **probability distribution** of train speeds and include this probability distribution in the model, leading to a *stochastic model*. The outputs of such a model would provide detailed information on the resulting probability distribution on possible durations.
- Finally, the speed could be taken into account as a **symbolic parameter** of the model. The result would be a *parametric model*, that would then yield a symbolic answer: the duration as a function of the speed of the train.

As one can see in Example 1, modeling with uncertainties can be done in several ways, the most general of which is the use of **non-determinism**. Indeed, non-determinism allows to represent complete lack of knowledge (but also lack of control) on a given system. Using non-determinism allows to incorporate a notion of choice inside the model, while having no information on how this choice is resolved. On the other hand, **probabilities** [1] represent some knowledge on the underlying choice: using probabilities allows to specify that there is a choice, and while we do not know what the outcome of this choice is, we have some information on how this outcome is chosen. Finally, the use of **parameters** allows to perform symbolic analysis, *i.e.*, producing results that are a function of those parameters, despite the uncertainty about their value. In this document, we will propose and study modeling formalisms that account for uncertainties in all these ways.

Although modeling with uncertainties yields more informative results than deterministic modeling (as can be seen from the example above), deterministic models are still widespread in most natural and social sciences. The main reasons for this fact are the lack of communication between formal sciences, which have been working on models with uncertainties for a long time, and those who might benefit from such models (see [DEB17] for an argumentation in the context of microbial modeling), but also the lack of efficient methods for analyzing large models with uncertainties. Indeed, since the aim of modeling is to pursue the analysis of the underlying system, some techniques and tools are necessary for performing such analysis.

The most common analysis method used in natural and social sciences is *simulation*, which consists in running a given model and observing its outputs. Simulation can be used both in the modeling and in the analysis phases: during modeling, simulation is used on intermediate models, and the outputs are compared to existing knowledge on the system (either in the form of

1. My personal belief – which might be debated, but this is out of the scope of this document – is that probabilities are just a mathematical concept that allows to abstract deterministic, but unknown, phenomena.

expertise or in the form of experimental data) in order to help the modeler refine its model. In the analysis phase, simulation is used on the final model in order to obtain predictions. In the case of deterministic systems, simulation is an appropriate technique as long as the outputs can be analyzed easily. However, when considering models with uncertainties, simulations need to be performed countless times in order to be able to observe a sufficient portion of the model's potential outputs. In this context, other alternative automated techniques are necessary. A substantial research effort – in particular in theoretical computer science – has been gathered during the last decades in order to develop such techniques. Among them lie *automated verification*.

Automated verification

Automated verification encompasses a multitude of techniques dedicated to analyzing models. The common ground between all those techniques is the need of having as inputs a *formal* model, *i.e.*, a model described in an unambiguous language with a well-defined semantics, and a set of formal properties, *i.e.*, queries that are also described in an unambiguous language and that the model/system must (or must not) satisfy. Model verification then consists in an automated (or semi-automated) analysis of the model to determine whether the given property is satisfied. Some verification techniques such as testing [MSB11] or statistical model checking [LDB10] are based on extensive simulation; Others, such as model checking [CJGK$^+$18] instead rely on an exhaustive exploration of the model. Finally, automated theorem-proving [Fit12] uses proof-theory along with the formal semantics of the model to prove (or disprove) that the model satisfies the given property. Whether they rely on simulation, exploration or proof-theory, automated verification techniques always provide more information than simple simulation. Indeed, a simple simulation only represents a single example of what the model can output. In the case of models with uncertainties, the potential outputs are numerous and therefore observing that a single output satisfies (or dissatisfies) the given property is not sufficient to conclude anything about the complete model. Instead, all automated verification techniques offer some guarantees about the obtained results: from percentage of covered test-cases in the context of testing to mathematical proofs for theorem-proving and model checking through error-rate and guaranteed precision for statistical model checking. Moreover, automated verification techniques also provide feedback on the results, mostly in terms of counter-examples when the property is not satisfied.

The unambiguity of the languages used to describe the model and the properties are requirements made necessary by the need for automation. Indeed, in order to develop tools that can

automatically analyze models, one needs to assume that the languages used to describe the model and the properties are well-defined, and can be automatically interpreted by a machine. A plethora of such languages (also called formalisms) have been developed over the years, from programming languages [KR+88, AGHH00, S+99, RV98] to more abstract specification formalisms [HMU01, Mil99, Abr05, Spi88], each of them dedicated to the description of certain types of systems [2]. Unfortunately as formalisms become more and more expressive (*i.e.*, capable of representing more and more detailed features of the systems), the accompanying verification techniques also raise more and more complexity, to the point that the automated verification of models expressed in some types of formalisms has been shown to be undecidable [ACD90, Esp94]. More research effort is therefore necessary to propose and study novel modeling formalisms and develop dedicated automated verification techniques. In this document, I will present some such contributions dedicated to the analysis of the behavior of discrete and continuous systems with uncertainty.

Contributions in the past 10 years

Since my Ph.D. thesis in 2010 [Del10], I have been interested in the complete scope of formal modeling and analysis of systems with uncertainties: from the theoretical study of modeling formalisms and verification techniques to the application of those theories to the analysis of concrete systems. Two main lines of works could in particular be observed in my thesis: (1) the study of a compositional specification theory for probabilistic systems called Constraint Markov Chains [CDL+11]; and (2) the development and use of a statistical verification technique – called statistical model checking – on concrete complex systems [BBB+10a].

2010-2013: Postdoctoral positions

During my years as a postdoctoral student, I have pursued these two lines of work. In the context of specification theories, I have enhanced the theory of Constraint Markov Chains to take into account non-determinism and modalities, yielding the formalism called Abstract Probabilistic Automata (APA) [DKL+11a, DKL+13]. This formalism has then been studied under all angles, yielding algebraic compositional results [DKL+11b, DLL13, DFLL13a, DLL14, DFLL14a] and a prototype tool for their verification [DLL+11a]. In the same line, I have also investigated other abstract formalisms for describing non-deterministic [BDF+13] and timed [DFH+12, DFLL13b,

2. The list is not exhaustive, the survey of all existing modeling formalisms is beyond the scope of this document.

DFLL14b, FLDL18] systems.
On the other hand, I have also pursued my work on statistical model checking, using this technique to verify different types of systems [BBB+12] and contributing to the development of a tool [BBD+12, NBB+15].

2013-present: Associate professor position

My main interests have not changed since I arrived in Nantes in 2013: my contributions can still be classified in the two lines presented above, *i.e.*, (1) the theoretical study of abstract modeling formalisms and analysis techniques, and (2) the development of practical verification techniques and tools and their application to concrete systems. However, the type of concrete systems I consider has slightly shifted. Indeed, interactions with several colleagues in the context of the ANR project PACS [3] have convinced me that *parameters* play an important role when modeling complex systems as they allow, in particular, symbolic analysis and robust synthesis. My main contributions since then have therefore been aimed at modeling and verifying (timed) probabilistic and parametric systems.

In the theoretical context, I have investigated several formalisms and techniques allowing to describe and analyze parametric systems. In particular, I have participated to the following contributions:
Probabilistic Event-B. From 2014 to 2017, I have contributed to the supervision of Mohamed Amine AOUADHI on the topic of Probabilistic Event-B. During this thesis, we have extended the Event-B formalism, which allows for the specification of parametric systems in a refinement-based fashion, to take into account probabilistic information [ADL17, ADL19b]. Besides enhancing the Event-B language and semantics to take into account probabilistic choice, we have also investigated the probabilistic refinement of probabilistic Event-B models as well as the automatic proving of some probabilistic properties.
Probabilistic time Petri nets. In 2016, we have proposed in [EDLR16] a new probabilistic syntax and semantics for Petri Nets, where, instead of using the standard stochastic semantics (*i.e.*, where probabilistic choice is resolved using races between concurrent transitions), we offer a discrete probabilistic choice that is independent of the timing information on transitions. Our aim with this formalism was to extend our results to the parametric context, where parameters can range both on the timing information on transitions and on the discrete probabilities. However, meaningful results in the parametric context still elude us.

3. https://anr.fr/Projet-ANR-14-CE28-0002

Parametric interval Markov chains. In 2015, I have proposed a first version of Interval Markov Chains where the interval endpoints can be replaced with parametric functions, along with first attempts to verifying meaningful properties such as consistency [Del15]. This line of work has been pursued with many co-authors, first yielding algorithms for parameter synthesis and verification of reachability properties [DLP16]. Between 2016 and 2017, I have then contributed to the supervision of Anicet BART on the adaptation of constraint programming techniques to this context in order to make the verification more efficient [BDL$^+$17, BDF$^+$18]. Notably, [BDL$^+$17] has been awarded the best paper award at the QEST conference in 2017.
In 2016, we have also extended the Parametric Interval Markov Chains formalism to take into account timing information, which has yielded Parametric Interval Probabilistic Timed Automata [AD16, ADF20]. In this context, we have shown that problems such as consistency and reachability are undecidable. Nevertheless, we have also provided semi-algorithms that allow to solve those problems under certain conditions. In these papers, parameters can only range on the timing information. Again, our intent was to have parameters ranging both on the timing information and probabilities, but the work is still ongoing.

On the other hand, I have have also pursued my work on the verification of concrete systems, by using (parametric) statistical techniques.
Parametric statistical model checking. Statistical model checking is a technique that has for a long time been limited to purely probabilistic systems. Indeed, because it relies on statistical sampling, it could not previously be used on systems where the transition probabilities were underspecified. In 2017, during the postdoctoral position of Paulin FOURNIER, we have used an adaptation of a technique called *importance sampling* in order to extend statistical model checking to parametric systems [DFL19]. This new technique has been implemented in a prototype tool and applied to a concrete industrial case study targeting the analysis of the flight plan of a UAV [4] [BAD$^+$19] during the thesis of Ran BAO, which I have contributed to supervise from 2017 to 2020.
Statistical model checking for parameter synthesis. Since 2016, I have developed a close collaboration with researchers in Oceanography (Lionel Guidy, LoV, Villefranche – Olivier Aumont, LOCEAN, Paris – Chris Bowler, IBENS, Paris) and Microbiology (Nicholas Bouskill, LBL, Berkeley), where I believe that some of the techniques I develop could be particularly useful. However, the first observation that we made was that those natural sciences mostly use deterministic modeling, despite of the benefits that would be gained by taking uncertainties into account. In 2017, we have therefore proposed an argument in favor of modeling with

4. Unmanned Aerial Vehicle

uncertainties in the context of Microbiology [DEB17]. We have also adapted a model taken from Oceanography in order to include uncertainties and used statistical model checking for synthesizing the optimal parameter values with respect to experimental data. This result, which has been published in 2020 in Scientific Reports [REG+20], a reknown generalist journal, has received praises from the Biology community, which is, to my opinion, a major achievement.

Automated learning. Finally, I have also considered the problem of automated modeling. Indeed, we would be very interested in some cases in using automated techniques for building models representing the statistical characteristics of some data sets while not necessarily building on the current knowledge of the underlying systems. To this purpose, I have contributed to the supervision of the thesis of Dimitri ANTAKLY from 2017-2020 on the development of automated learning and verification techniques for security assessment. During his thesis, we have developed *learning techniques* for a recent formalism called *Recursive Timescale Graphical Event Models* along with verification techniques allowing to select a model that is the most representative of the input data while satisfying some given properties [ADL19a]. This new method has for now only been applied to security assessment of software data, but our intent is to port it to the Oceanography context in the near future.

Outline of the document

This document relates four of the contributions presented above. Each of them was obtained during the Ph.D. thesis / postdoc of some of my students. In order to be representative of my research axes, I have chosen to present two contributions on the theoretical side (Chapters 1 and 2) and two contributions on the practical side (Chapters 3 and 4).

— Chapter 1 introduces and studies parametric Interval Markov chains and presents our adaptation of constraint programming methods for the synthesis of parameters values ensuring properties such as consistency or reachability.

— In Chapter 2, I present our extension of the Event-B method in order to replace some of the non-deterministic or parametric choices with probabilities.

— Chapter 3 details our adaptation of the statistical model checking technique to the parametric context, then introduces our prototype tool and presents its application to an industrial case study.

— Chapter 4 presents our study of the Recursive Timed Graphical Event Model formalism along with automated learning and verification techniques.

— Finally, the last Chapter presents perspectives for my future research topics.

In addition to the chapters presented above, this document contains appendices. In particular, Appendix B gathers the main notations and definitions introduced in each chapter. It is aimed to be detached from the manuscript so the reader can use it while reading the main part of the document.

CHAPTER 1

PARAMETRIC INTERVAL MARKOV CHAINS

The work we present in this chapter was mostly achieved during **Anicet Bart**'s thesis, although it follows a line of work initiated more than 10 years ago on specification theories for probabilistic systems, which has yielded numerous publications on several types of models: Modal Transition Systems [BDF+13], Interval Markov Chains [DLL+11b, DLL+12a], Constraint Markov Chains [CDL+10, CDL+11, DLL+12b] or Abstract Probabilistic Automata [DKL+11a, DKL+11b, DLL+11a, DLL13, DFLL13a, DKL+13, DLL14, DFLL14a]. The work on parametric models was initiated in 2015, motivated by the start of the ANR **PACS** project, and has also yielded several publications [Del15, DLP16, BDL+17, BDF+18] leading to results presented here. It is worth noting that the results presented in this chapter have been awarded the best paper award at the QEST conference in 2017.

This chapter focuses on systems where uncertainty comes from several sources. The first source is the probabilistic nature of the systems themselves (Markov Chains). The second source is the underspecification of those systems: instead of considering probabilistic transitions as usually done in Markov Chains, we consider here intervals of probabilities on the transitions, therefore leading to a *specification* formalism. In other word, each Interval Markov Chain we analyze is an abstract representation of a familly of systems. Finally, the last source of uncertainty is a second level of underspecification: the endpoints of the probabilistic intervals might not be known precisely, and are therefore replaced with symbolic parameters. While we had already studied parametric Interval Markov Chains in [Del15, DLP16], this chapter presents an alternative and more efficient solution to several problems of interest already solved in [DLP16], using techniques that are adapted from Constraint Programming. For the sake of conciseness, most of the proofs of the results presented in this chapter have been left out of this document. The interested reader can find them in [BDF+18].

1.1 Introduction

Discrete time Markov chains (MCs for short) are a standard probabilistic modeling formalism that has been extensively used in the litterature to reason about software [WT94] and real-life systems [HDR10]. However, when modeling real-life systems, the exact value of transition probabilities may not be known precisely. Several formalisms abstracting MCs have therefore been developed. Parametric Markov chains [AHV93] (pMCs for short) extend MCs by allowing parameters to appear in transition probabilities. In this formalism, parameters are variables and transition probabilities may be expressed as polynomials or rational functions over these variables. A given pMC represents a potentially infinite set of MCs, obtained by replacing each parameter by a given value. pMCs are particularly useful to represent systems where dependencies between transition probabilities are required. Indeed, a given parameter may appear in several distinct transition probabilities, which requires that the same value is given to all its occurences. Interval Markov chains [JL91] (IMCs for short) extend MCs by allowing precise transition probabilities to be replaced by intervals, but cannot represent dependencies between distinct transitions. IMCs have mainly been studied with three distinct semantics interpretations. Under the *once-and-for-all* semantics, a given IMC represents a potentially infinite number of MCs where transition probabilities are chosen inside the specified intervals while keeping the same underlying graph structure. The *interval-Markov-decision-process* semantics (IMDP for short), such as presented in [CSH08, SVA06], does not require MCs to preserve the underlying graph structure of the original IMC but instead allows a finite "unfolding" of the original graph structure: new probability values inside the intervals can be chosen each time a state is visited. Finally, the *at-every-step* semantics, which was the original semantics given to IMCs in [JL91], does not preserve the underlying graph structure too while allowing to "aggregate" and "split" states of the original IMC in the manner of probabilistic simulation.

Model checking algorithms and tools have been developed in the context of pMCs [DJJ$^+$15, HHWZ10, KNP11] and IMCs with the once-and-for-all and the IMDP semantics [CK15, BLW13]. State of the art tools [DJJ$^+$15] for pMC verification compute a rational function on the parameters that characterizes the probability of satisfying a given property, and then use external tools such as SMT solving [DJJ$^+$15] for computing the satisfying parameter values. For these methods to be viable in practice, the allowed number of parameters is quite limited. On the other hand, the model checking procedure for IMCs presented in [BLW13] is adapted from machine learning and builds successive refinements of the original IMCs that optimize the probability of satisfying the given property. This algorithm converges, but not necessarily

to a global optimum. It is worth noticing that existing model checking procedures for pMCs and IMCs strongly rely on their underlying graph structure (*i.e.*, respect the once-and-for-all semantics). However, in [CSH08] the authors perform model checking of ω-PCTL formulas on IMCs w.r.t. the IMDP semantics and they show that model checking of LTL formulas can be solved for the IMDP semantics by reduction to the model checking problem of ω-PCTL on IMCs with the IMDP semantics. For all that, to the best of our knowledge, no solutions for model checking IMCs with the at-every-step semantics have been proposed yet.

In this chapter, we focus on Parametric interval Markov chains [DLP16] (pIMCs for short), that generalize both IMCs and pMCs by allowing parameters to appear in the endpoints of the intervals specifying transition probabilities, and summarize the contributions obtained during Anicet Bart's thesis. Details and proofs of these contributions can be found in [BDF$^+$18, BDL$^+$17]. The main results are the following.

First, we formally compare abstraction formalisms for MCs in terms of succinctness: we show in particular in Proposition 2 that pIMCs are *(strictly) more succinct* than both pMCs and IMCs when equipped with the right semantics. In other words, everything that can be expressed using pMCs or IMCs can also be expressed using pIMCs while the reverse does not always hold.

Second, we prove in Theorem 1 that the once-and-for-all, the IMDP, and the at-every-step semantics are equivalent w.r.t. reachability properties, both in the IMC and in the pIMC settings. Notably, this result gives theoretical backing to the generalization of existing works on the verification of IMCs to the at-every-step semantics.

Third, we study the parametric verification of fundamental properties at the pIMC level: consistency, qualitative reachability, and quantitative reachability. Given the expressivity of the pIMC formalism, the risk of producing a pIMC specification that is incoherent and therefore does not model any concrete MC is high. Consistency is therefore of paramount importance. In order to provide solutions to consistency and reachability, we model these problems using constraint encodings and use state of the art constraint solvers for solving them. Constraints are first order logic predicates used for modeling and solving combinatorial problems [RBW06]. A problem is described with a list of variables, each in a given domain of possible values, together with a list of constraints over these variables. Such problems are then sent to solvers which decide whether the problem is satisfiable, *i.e.*, if there exists a valuation of the variables satisfying all constraints, and in this case computes a solution. We therefore propose, in Section 1.4.1 and Proposition 3, constraint encodings for deciding whether a given pIMC is consistent and, if so, synthesizing parameter values ensuring consistency. We then extend, in Section 1.4.2 and Proposition 4, these encodings to qualitative reachability, *i.e.*, ensuring that given state labels are reachable in *all*

(resp. *none*) of the MCs modeled by a given pIMC. Finally, in Section 1.5.2 and Theorem 2, we focus on the quantitative reachability problem, *i.e.*, synthesizing parameter values such that the probability of reaching given state labels satisfies fixed bounds in *at least one* (resp. *all*) MCs modeled by a given pIMC. While consistency and qualitative reachability for pIMCs have already been studied in [DLP16], the constraint encodings we propose are significantly smaller (linear instead of exponential). To the best of our knowledge, our results have provided the first solution to the quantitative reachability problem for pIMCs.
The last contribution, presented in Section 1.6, is the implementation of all our verification algorithms in a prototype tool that generates the required constraint encodings and can be plugged to any SMT solver for their resolution.

1.2 Background

We start by introducing notions and notations that will be used throughout this chapter. Given a finite set of variables $X = \{x_1, \ldots, x_k\}$ we call *domain* a set of values associated to a variable in X. We write D_x for the domain associated to the variable $x \in X$ and D_X for the set of domains associated to the variables in X. A *valuation* v over X is a set $v = \{(x,d) \mid x \in X, d \in D_x\}$ of elementary valuations (x,d) where for each $x \in X$ there exists a unique pair of the form (x,d) in v. When clear from the context, we write $v(x) = d$ for the value given to variable x according to valuation v. A *rational function* f over X is a division of two (multivariate) polynomials g_1 and g_2 over X with rational coefficients, *i.e.*, $f = g_1/g_2$. We write $\mathbb{Q}$ for the set of rational numbers and $\mathbb{Q}_X$ for the set of rational functions over X. The evaluation $v(g)$ of a polynomial g under the valuation v replaces each variable $x \in X$ by its value $v(x)$.
An *atomic constraint* over X is a Boolean expression of the form $f(X) \bowtie g(X)$, with $\bowtie \in \{\leq, \geq, <, >, =\}$ and f and g two functions over variables in X. An atomic constraint is *linear* if the functions f and g are linear. A *constraint* over X is a Boolean combination of atomic constraints over X.
Given a finite set of states S, we write $\mathsf{Dist}(S)$ for the set of *probability distributions* over S, *i.e.*, the set of functions $\mu\colon S \to [0,1]$ such that $\sum_{s \in S} \mu(s) = 1$. We write $\mathbb{I}$ for the set containing all interval subsets of $[0,1]$. In the following, we consider a universal set of symbols A that we use for labelling the states of our structures. We call these symbols *atomic propositions*. We will use Latin alphabet in state context and Greek alphabet in atomic proposition context.

Constraints. *Constraints* are first order logic predicates over a given set of variables. Informally,

a *Constraint Satisfaction Problem* (CSP) consists of a set of variables associated to given domains and subject to a set of constraints. A CSP is *satisfiable* if there exists a valuation of the variables that satisfies all constraints. Checking satisfiability of constraint problems is difficult in general, as the space of all possible valuations has a size exponential in the number of variables.

Definition 1 (Constraint Satisfaction Problem)**.** A *Constraint Satisfaction Problem* (CSP) is a tuple $\Omega = (X, D, C)$ where X is a finite set of variables, $D = D_X$ is the set of all domains associated to the variables from X, and C is a set of constraints over X.

We say that a valuation over X satisfies Ω if and only if it satisfies all constraints in C. We write $v(C)$ for the satisfaction result of the valuation of the constraints C according to v (*i.e.*, true or false). In the following we call CSP *encoding* a scheme for formulating a given problem into a CSP. The size of a CSP corresponds to the number of variables and atomic constraints appearing in the problem. Note that, in constraint programming, having less variables or less constraints during the encoding does not necessarily imply faster solving time of the problems.

Discrete Time Markov Chains.

Definition 2 (Discrete Time Markov Chain)**.** A Discrete Time Markov Chain (MC for short) is a tuple $\mathcal{M} = (S, s_0, p, L)$, where S is a finite set of states containing the initial state s_0, $L\colon S \to 2^A$ is a labelling function, and $p\colon S \times S \to [0,1]$ is a probabilistic transition function such that for all $s \in S$, the function $p(s,.)$ is a distribution (*i.e.*, $s' \mapsto p(s,s') \in \mathsf{Dist}(S)$).

The labelling function L of a MC $\mathcal{M} = (S, s_0, p, L)$ associates to each state $s \in S$ a set of atomic propositions. Given a state $s \in S$, $L(s)$ is called the *label* of s. We write $\mathtt{MC}$ for the set containing all discrete time Markov chains.
A Markov Chain can be represented as a directed graph where the nodes correspond to the states of the MC and the edges are labelled with the probabilities given by the transition function of the MC. In this representation, a missing transition between two states represents a transition probability of zero. As usual, given a MC $\mathcal{M}$, we call a *run* of $\mathcal{M}$ a sequence of states obtained from executing $\mathcal{M}$, *i.e.*, a sequence $w = s_1, s_2, \ldots$ such that the probability of taking the transition from s_i to s_{i+1} is strictly positive: $p(s_i, s_{i+1}) > 0$, for all i. A run w is finite if and only if it belongs to S^*, *i.e.*, it represents a finite sequence of transitions from $\mathcal{M}$.

Example 2. Figure 1.1 illustrates the MC $\mathcal{M}_1 = (S, s_0, p, L) \in \mathtt{MC}$ where the set of states S is given by $\{s_0, s_1, s_2, s_3, s_4\}$, the atomic propositions are restricted to $\{\alpha, \beta\}$, the initial state is s_0, and the labelling function L is as follows: $\{(s_0, \varnothing), (s_1, \{\alpha\}), (s_2, \{\beta\}), (s_3, \{\alpha, \beta\}), (s_4, \{\alpha\})\}$.

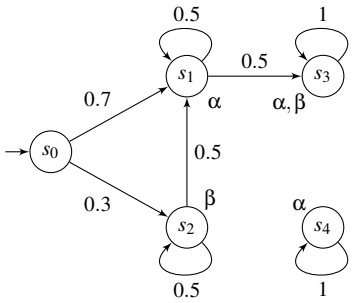


Figure 1.1 – MC $\mathcal{M}_1$

The sequences of states $w = s_0, s_1, s_2$, $w' = s_0, s_2$, and $w'' = s_0, s_2, s_2, s_2$ are three (finite) runs from the initial state s_0 to the state s_2.

In order to prove some of our results, we need a notion of bisimulation between Markov chains. We therefore use the classical notion of probabilistic bisimulation.

Definition 3 (Probabilistic bisimulation for Markov chains [BK08])**.** Let $\mathcal{M} = (S, s_0, p, L)$ be a Markov chain. A *probabilistic* bisimulation on $\mathcal{M}$ is an equivalence relation $\mathcal{R}$ on S such that for all pair of states $(s_1, s_2) \in \mathcal{R}$, we have

— $L(s_1) = L(s_2)$, and

— $p(s_1, T) = p(s_2, T)$ for each equivalence class $T \in S/\mathcal{R}$.

Given two MCs $\mathcal{M}_1$ and $\mathcal{M}_2$, we say that $\mathcal{M}_1$ and $\mathcal{M}_2$ are *bisimilar* if and only if there exists a probabilistic bisimulation $\mathcal{R}$ on $\mathcal{M}_1 \cup \mathcal{M}_2$ with $(s_0^1, s_0^2) \in \mathcal{R}$, where $\mathcal{M}_1 \cup \mathcal{M}_2$ is a MC consisting of all states, transitions and labels of $\mathcal{M}_1$ and $\mathcal{M}_2$ [1].

Reachability. A Markov chain $\mathcal{M}$ defines a unique probability measure $\mu^{\mathcal{M}}$ over the set of infinite sequences $s_0, s_1, \ldots \in S^\omega$ (see [BK08] for details). According to this measure, the probability of a *finite* sequence of states $w = s_0, s_1, \ldots, s_n$, written $\mathbb{P}_{\mathcal{M}}(w)$ is the product of the probabilities of the transitions transitions involved in this sequence, *i.e.*, $\mathbb{P}_{\mathcal{M}}(w) = p(s_0, s_1) \cdot p(s_1, s_2) \cdot \ldots \cdot p(s_{n-1}, s_n)$. Naturally, if w is not a run of $\mathcal{M}$, we have $\mathbb{P}_{\mathcal{M}}(w) = 0$.
Given a MC $\mathcal{M}$, the overall probability of reaching a given state s from the initial state s_0 is called the *reachability probability* and written $\mathbb{P}^{s_0}_{\mathcal{M}}(\Diamond s)$ or $\mathbb{P}_{\mathcal{M}}(\Diamond s)$ when clear from the context. This probability is computed as the sum of the probabilities of all finite runs starting in the initial state and reaching the state s for the first time. Formally, let $\mathsf{reach}_{s_0}(s) =$

1. The initial state is not important here, so it could be either s_0^1 or s_0^2 w.l.o.g.

$\{ w \in S^* \mid w = s_0, \dots s_n$ with $s_n = s$ and $s_i \neq s \ \forall 0 \leq i < n \}$ be the set of such runs. We then define $\mathbb{P}_{\mathcal{M}}(\Diamond s) = \sum_{w \in \mathsf{reach}_{s_0}(s)} \mathbb{P}_{\mathcal{M}}(w)$ if $s \neq s_0$ and 1 otherwise. This notation naturally extends to the reachability probability of a state s from a state t that is not s_0, written $\mathbb{P}^t_{\mathcal{M}}(\Diamond s)$ and to the probability of reaching a state label $\Gamma \subseteq A$ written $\mathbb{P}^{s_0}_{\mathcal{M}}(\Diamond \Gamma)$. One could also extend it to the reachability probability of an atomic proposition α by considering all state labels containing α. In the following, we say that a state s (resp. a label $\Gamma \subseteq A$) is reachable in $\mathcal{M}$ if and only if the reachability probability of this state (resp. label) from the initial state is strictly positive.

Example 3 (Example 2 continued)**.** In Figure 1.1 the probability of the run $w = s_0, s_2, s_1, s_1, s_3$ is $0.3 \cdot 0.5 \cdot 0.5 \cdot 0.5 = 0.0375$ and the probability of reaching the state s_1 from s_0 is $\mathbb{P}^{s_0}_{\mathcal{M}_1}(\Diamond s_1) = p(s_0,s_1) + \Sigma_{i=0}^{+\infty} p(s_0,s_2) \cdot p(s_2,s_2)^i \cdot p(s_2,s_1) = p(s_0,s_1) + p(s_0,s_2) \cdot p(s_2,s_1) \cdot (1/(1 - p(s_2,s_2))) = 1$. Furthermore, the probability of reaching $\{\beta\}$ corresponds to the probability of reaching the state s_2, which is 0.3 here.

1.3 Markov chain abstractions

Modeling an application as a Markov Chain requires knowing the exact probability for each possible transition of the system. However, this can be difficult to compute or to measure in the case of a real-life application (*e.g.*, because of precision errors or limited knowledge). In this section, we start with a generic definition of Markov chain abstraction models. Then we recall three abstraction models from the literature, respectively pMC, IMC, and pIMC, and finally we present a comparison of these existing models in terms of succinctness.

Definition 4 (Markov chain Abstraction Model)**.** A Markov chain abstraction model (an abstraction model for short) is a pair $(\mathtt{L}, \models)$ where $\mathtt{L}$ is a nonempty set of models and $\models$ is a relation between $\mathtt{MC}$ and $\mathtt{L}$. Let $\mathcal{P}$ be in $\mathtt{L}$ and $\mathcal{M}$ be in $\mathtt{MC}$. We say that $\mathcal{M}$ implements $\mathcal{P}$ if and only if $(\mathcal{M}, \mathcal{P})$ belongs to $\models$ (*i.e.*, $\mathcal{M} \models \mathcal{P}$). When the context is clear, we do not mention the satisfaction relation $\models$ and only use $\mathtt{L}$ to refer to the abstraction model $(\mathtt{L}, \models)$.

A *Markov chain abstraction model* is a specification theory for MCs. It consists of a set of abstract models, called *specifications*, each of which representing a (potentially infinite) set of MCs – *implementations* – together with a satisfaction relation defining the link between implementations and specifications. As an example, consider the powerset of $\mathtt{MC}$ (*i.e.*, the set containing all possible sets of Markov chains). Clearly, $(2^{\mathtt{MC}}, \in)$ is a Markov chain abstraction model, which we call the *canonical abstraction model*. This abstraction model has the advantage of representing all possible sets of Markov chains but it also has the disadvantage that some

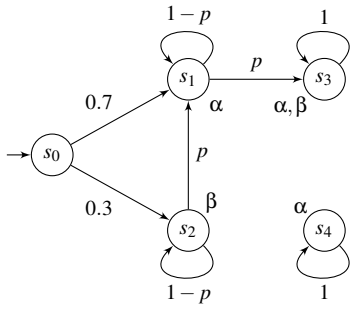


Figure 1.2 – pMC I'

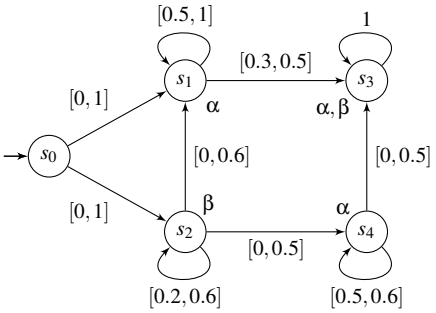


Figure 1.3 – IMC I

Markov chain abstractions are only representable by an infinite extension representation. Indeed, recall that there exist subsets of $[0,1] \subseteq \mathbb{R}$ which cannot be represented in a finite space (e.g., the Cantor set [Can83]). We now present existing MC abstraction models from the literature.

1.3.1 Existing MC abstraction models

Parametric Markov Chain is a MC abstraction model from [AHV93] where a transition can be annotated by a rational function over *parameters*. We write pMC for the set containing all parametric Markov chains.

Definition 5 (Parametric Markov Chain)**.** A Parametric Markov Chain (pMC for short) is a tuple $I = (S, s_0, P, L, \mathbb{X})$ where S, s_0, and L are defined as for MCs, $\mathbb{X}$ is a set of variables (parameters) ranging over $[0,1]$, and $P\colon S \times S \to \mathbb{Q}_{\mathbb{X}}$ associates with each potential transition a parameterized probability.

Let $\mathcal{M} = (S, s_0, p, L)$ be a MC and $I = (S', s'_0, P, L', \mathbb{X})$ be a pMC. The satisfaction relation $\models_{\mathrm{p}}$ between MC and pMC is defined by $\mathcal{M} \models_{\mathrm{p}} I$ if and only if $S = S'$, $s_0 = s'_0$, $L = L'$, and there exists a valuation v of $\mathbb{X}$ such that $p(s,s')$ equals $v(P(s,s'))$ for all s, s' in S.

Example 4. Figure 1.2 shows a pMC $I' = (S, s_0, P, L, \mathbb{X})$ where S, s_0, and L are identical to those of the MC $\mathcal{M}$ from Figure 1.1, the set $\mathbb{X}$ contains only one variable p, and the parametric transitions in P are given by the edge labelling (*e.g.*, $P(s_0, s_1) = 0.7$, $P(s_1, s_3) = p$, and $P(s_2, s_2) = 1 - p$). Note that the pMC I' is a specification containing the MC $\mathcal{M}_1$ from Figure 1.1 (using the valuation v such that $v(p) = 0.5$).

Interval Markov Chains extend MCs by allowing to label transitions with intervals of possible probabilities instead of precise probabilities. We write IMC for the set containing all interval Markov chains.

Definition 6 (Interval Markov Chain [JL91])**.** An Interval Markov Chain (IMC for short) is a tuple $I = (S, s_0, P, L)$, where S, s_0, and L are defined as for MCs, and $P\colon S \times S \to \mathbb{I}$ associates with each potential transition an interval of probabilities.

Example 5. Figure 1.3 illustrates an IMC $I = (S, s_0, P, L)$ where S, s_0, and L are similar to the MC given in Figure 1.1. By observing the edge labelling we see that $P(s_0, s_1) = [0,1]$, $P(s_1, s_1) = [0.5, 1]$, and $P(s_3, s_3) = [1,1]$. On the other hand, the intervals of probability for missing transitions are reduced to $[0,0]$, e.g., $P(s_0, s_0) = [0,0]$, $P(s_0, s_3) = [0,0]$, $P(s_1, s_4) = [0,0]$.

In the literature, IMCs have been mainly used with three distinct semantics: *at-every-step*, *interval-Markov-decision-process*, and *once-and-for-all*. All these semantics are associated with distinct satisfaction relations which we now introduce.
The *once-and-for-all* IMC semantics [DJJ$^+$15, WTO$^+$11, PLSVS13] is alike to the semantics for pMC, as introduced above. The associated satisfaction relation $\models^o_{\mathbb{I}}$ is defined as follows.

Definition 7 (Once-and-for-all satisfaction)**.** A MC $\mathcal{M} = (T, t_0, p, L^M)$ satisfies an IMC $I = (S, s_0, P, L^I)$ with the once-and-for-all semantics, written $\mathcal{M} \models^o_{\mathbb{I}} I$, if and only if $(T, t_0, L^M) = (S, s_0, L^I)$ and for all reachable states s in $\mathcal{M}$ and all state $s' \in S$, $p(s, s') \in P(s, s')$.

In this sense, we say that MC implementations using the once-and-for-all semantics need to have the same structure as the IMC specification. Remark that this definition only targets *reachable* states in the MC. Indeed, some states of the IMC could have inconsistent outgoing transitions. While these states cannot be "satisfied" in MC implementations, satisfaction should still be possible if they are unreachable.
Next, the *interval-Markov-decision-process* IMC semantics (IMDP for short) [CSH08, SVA06] operates as an "unfolding" of the original IMC by picking each time a state is visited a possibly new probability distribution which respects the intervals of probabilities. Thus, this semantics allows one to produce MCs satisfying IMCs with a different structure. Formally, the associated satisfaction relation $\models^d_{\mathbb{I}}$ is defined as follows.

Definition 8 (Interval-Markov-decision-process satisfaction)**.** A MC $\mathcal{M} = (T, t_0, p, L^M)$ satisfies an IMC $I = (S, s_0, P, L^I)$ with the IMDP semantics, written $\mathcal{M} \models^d_{\mathbb{I}} I$, if and only if there exists a mapping π from T to S such that

1. $\pi(t_0) = s_0$,
2. $L^I(\pi(t)) = L^M(t)$ for all states $t \in T$, and
3. $p(t, t') \in P(\pi(t), \pi(t'))$ for all pairs of states t, t' in T, where t is reachable in $\mathcal{M}$.

Thus, we have that $\models^d_{\mathtt{I}}$ is more general than $\models^o_{\mathtt{I}}$ (*i.e.*, whenever $\mathcal{M} \models^o_{\mathtt{I}} I$ we also have $\mathcal{M} \models^d_{\mathtt{I}} I$ for the identity mapping π). Note that in [CSH08, SVA06] the authors allow the Markov chains satisfying the IMCs w.r.t. $\models^d_{\mathtt{I}}$ to have an infinite state space. In this work we consider Markov chains with a finite state space only, therefore MC implementations with respect to the IMDP semantics consist in a finite unfolding of the specification IMC, which corresponds to a transient IMDP semantics followed by a once-and-for-all semantics.
Finally, the *at-every-step* IMC semantics, first introduced in [JL91], operates as a simulation relation based on the transition probabilities and state labels, and therefore allows MC implementations to have a different structure than the IMC specification. Compared to the previous semantics, in addition to the unfoldings this one allows to "aggregate" and "split" states from the original IMC. Formally, the associated satisfaction relation $\models^a_{\mathtt{I}}$ is defined as follows.

Definition 9 (At-every-step satisfaction)**.** A MC $\mathcal{M} = (T, t_0, p, L^M)$ satisfies an IMC $I = (S, s_0, P, L^I)$ with the at-every-step semantics, written $\mathcal{M} \models^a_{\mathtt{I}} I$ if and only if there exists a relation $\mathcal{R} \subseteq T \times S$ such that $(t_0, s_0) \in \mathcal{R}$, and whenever $(t, s) \in \mathcal{R}$, we have

1. the labels of s and t correspond: $L^M(t) = L^I(s)$,
2. there exists a correspondence function $\delta_{(t,s)} : T \to (S \to [0,1])$ such that
   (a) $\forall t' \in T$ if $p(t,t') > 0$ then $\delta_{(t,s)}(t')$ is a distribution on S
   (b) For all $s' \in S$, $(\Sigma_{t' \in T} p(t,t') \cdot \delta_{(t,s)}(t')(s')) \in P(s,s')$, and
   (c) For all $(t',s') \in T \times S$, if $\delta_{(t,s)}(t')(s') > 0$, then $(t',s') \in \mathcal{R}$.

Remark that reachability of the states from $\mathcal{M}$ is not required for the at-every-step satisfaction relation. Indeed, in this case, states that are not reachable in $\mathcal{M}$ can be absent from the satisfaction relation, and therefore do not impact at-every-step satisfaction.
Example 6 illustrates the three IMC semantics and Proposition 1 compares them. We say that an IMC semantics $\models_1$ is more general than another IMC semantics $\models_2$ if and only if for all IMCs I and for all MCs $\mathcal{M}$ if $\mathcal{M} \models_2 I$ then $\mathcal{M} \models_1 I$. Also, $\models_1$ is strictly more general than $\models_2$ if and only if $\models_1$ is more general than $\models_2$ and $\models_2$ is not more general than $\models_1$.

Example 6 (Example 5 continued)**.** Consider the IMC I from Figure 1.3, the MC $\mathcal{M}_1$ from Figure 1.1, the MC $\mathcal{M}_2$ from Figure 1.4, and the MC $\mathcal{M}_3$ from Figure 1.5. We have that $\mathcal{M}_1$ satisfies I w.r.t. $\models^o_{\mathtt{I}}$ and we say that $\mathcal{M}_1$ has the same structure as I. Trivially, we also have that $\mathcal{M}_1$ satisfies I w.r.t. $\models^d_{\mathtt{I}}$ and $\models^a_{\mathtt{I}}$.
Regarding $\mathcal{M}_2$, note that two probability distributions have been chosen for the state s_1 from I. This produces two states t_1 and t'_1 in $\mathcal{M}_2$ and changes the structure of the graph. Thus, $\mathcal{M}_2 \not\models^o_{\mathtt{I}} I$.

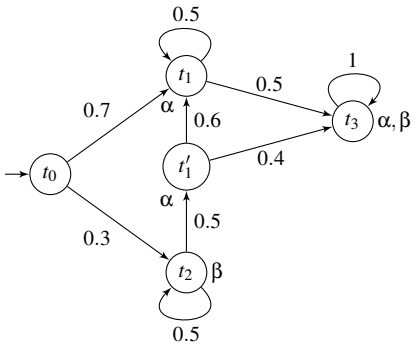


Figure 1.4 – MC $\mathcal{M}_2$ satisfying the IMC I from Figure 1.3 w.r.t. $\models^d_{\mathtt{I}}$

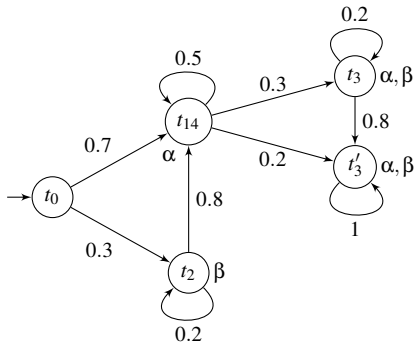


Figure 1.5 – MC $\mathcal{M}_3$ satisfying the IMC I from Figure 1.3 w.r.t. $\models^a_{\mathtt{I}}$

On the other hand, we have that $\mathcal{M}_2$ satisfies I w.r.t. $\models^d_{\mathtt{I}}$ with the mapping $\pi(t_i) = s_i$ for all t_i and $\pi(t'_1) = s_1$. Remark that there is no state t_4 here, but having an unreachable state t_4 and defining $\pi(t_4) = s_4$ would not prevent satisfaction. Trivially, we also have $\mathcal{M}_2 \models^a_{\mathtt{I}} I$ with the relation $\mathcal{R}_2 = \{\,(t,s) \mid \pi(t) = s\,\}$.

Finally, in the MC $\mathcal{M}_3$ with state space T the state s_3 from I has been "split" into two states t_3 and t'_3 and the state t_{14} "aggregates" the states s_1 and s_4 from I. The relation $\mathcal{R}_3 \subseteq T \times S$ containing the pairs (t_0,s_0), (t_{14},s_1), (t_{14},s_4), (t_2,s_2), (t_3,s_3), and (t'_3,s_3) is a satisfaction relation between $\mathcal{M}_3$ and I such as defined by $\models^a_{\mathtt{I}}$. For instance, consider the pair $(t_2,s_2) \in \mathcal{R}$. For this pair, define the correspondence function $\delta_{(t_2,s_2)}$ such that $\delta_{(t_2,s_2)}(t_2)(s_2) = 1$, $\delta_{(t_2,s_2)}(t_{14})(s_1) = 0.5$, $\delta_{(t_2,s_2)}(t_{14})(s_4) = 0.5$, and $\delta_{(t_2,s_2)}(t')(s') = 0$ otherwise. We can verify the following:

1. For all t' such that $p_2(t_2,t') > 0$, $\delta_{(t_2,s_2)}(t')$ is a distribution on S. This is the case for t_2 and t_{14}.
2. For all $s' \in S$, $(\Sigma_{t' \in T_2} p_2(t_2,t') \cdot \delta_{(t_2,s_2)}(t')(s')) \in P(s_2,s')$:
   $[\mathbf{s_1}]: \quad (\Sigma_{t' \in T_2} p_2(t_2,t') \cdot \delta_{(t_2,s_2)}(t')(s_1)) = 0.8 * 0.5 = 0.4 \in [0,0.6] = P(s_2,s_1)$
   $[\mathbf{s_2}]: \quad (\Sigma_{t' \in T_2} p_2(t_2,t') \cdot \delta_{(t_2,s_2)}(t')(s_2)) = 0.2 * 1 = 0.2 \in [0.2,0.6] = P(s_2,s_2)$
   $[\mathbf{s_4}]: \quad (\Sigma_{t' \in T_2} p_2(t_2,t') \cdot \delta_{(t_2,s_2)}(t')(s_4)) = 0.8 * 0.5 = 0.4 \in [0,0.5] = P(s_2,s_4)$
3. For all $(t',s') \in T_2 \times S$, if $\delta_{(t_2,s_2)}(t')(s') > 0$, then $(t',s') \in \mathcal{R}_2$. This is the case for (t_2,s_2), (t_{14},s_1), and (t_{14},s_4).

For all other pairs $(t',s') \in \mathcal{R}_2$, the correspondence functions can be defined trivially as Dirac distributions where $\delta_{(t',s')}(t_{14})$ is either a Dirac on s_1, for $(t',s') = (t_0,s_0)$ and (t_{14},s_1), or a Dirac on s_4 for $(t',s') = (t_{14},s_4)$.

Thus, $\mathcal{M}_3 \models^a_{\mathtt{I}} I$. On the other hand, $\mathcal{M}_3 \not\models^d_{\mathtt{I}} I$ since there exist probabilities on transitions that cannot belong to their respective interval of probabilities on the IMC (*e.g.*, $p(t_2,t_{14}) = 0.8 \notin$

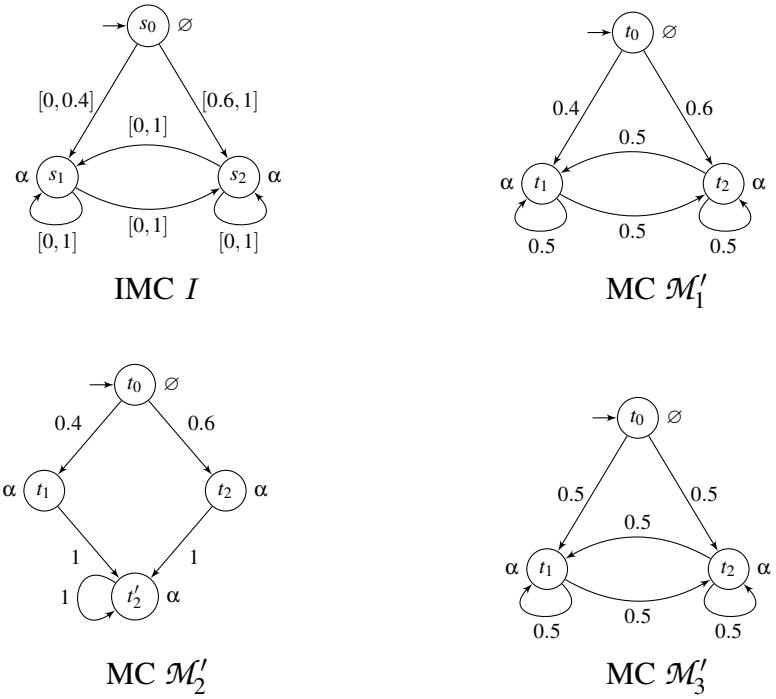


Figure 1.6 – IMC I, MCs $\mathcal{M}'_1$, $\mathcal{M}'_2$, and $\mathcal{M}'_3$ such that $\mathcal{M}'_1 \models^a_{\mathtt{I}} I$, $\mathcal{M}'_1 \models^d_{\mathtt{I}} I$ and $\mathcal{M}'_1 \models^o_{\mathtt{I}} I$; $\mathcal{M}'_2 \models^d_{\mathtt{I}} I$ and $\mathcal{M}'_2 \not\models^o_{\mathtt{I}} I$; $\mathcal{M}'_3 \models^a_{\mathtt{I}} I$ and $\mathcal{M}'_3 \not\models^d_{\mathtt{I}} I$.

$[0, 0.6] = P(s_2, s_1)$).

Proposition 1. *The* at-every-step *satisfaction relation is (strictly) more general than the* interval-Markov-decision-process *satisfaction relation which is (strictly) more general than the* once-and-for-all *satisfaction relation.*

Examples illustrating this result are given in Figure 1.6 and a complete proof is provided in [BDF+18].

Parametric Interval Markov Chains, as introduced in [DLP16], abstract IMCs by allowing (combinations of) parameters to be used as interval endpoints in IMCs. Under a given parameter valuation the pIMC yields an IMC as introduced above. pIMCs therefore allow the representation, in a compact way and with a finite structure, of a potentially infinite number of IMCs. Note that one parameter can appear in several transitions at once, requiring the associated transition probabilities to depend on one another. Let $\mathbb{X}$ be a finite set of parameters and v be a valuation over $\mathbb{X}$. By combining notations used for IMCs and pMCs the set $\mathbb{I}(\mathbb{Q}_{\mathbb{X}})$ contains all parametrized intervals over $[0,1]$, and for all $J = [f_1, f_2] \in \mathbb{I}(\mathbb{Q}_{\mathbb{X}})$, $v(J)$ denotes the interval $[v(f_1), v(f_2)]$ if

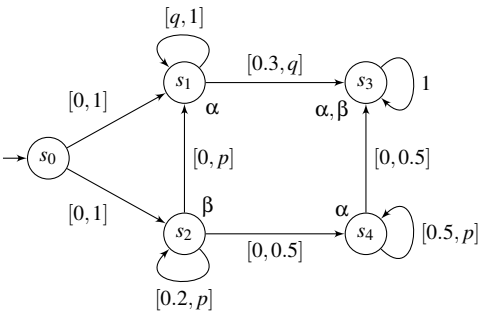


Figure 1.7 – pIMC $\mathcal{P}$

$0 \leq v(f_1) \leq v(f_2) \leq 1$ and the empty set otherwise [2]. We write $\texttt{pIMC}$ for the set containing all parametric interval Markov chains.

Definition 10 (Parametric Interval Markov Chain [DLP16])**.** A Parametric Interval Markov Chain (pIMC for short) is a tuple $\mathcal{P} = (S, s_0, P, L, \mathbb{X})$, where S, s_0, L, and $\mathbb{X}$ are defined as for pMCs, and $P\colon S \times S \to \mathbb{I}(\mathbb{Q}_{\mathbb{X}})$ associates with each potential transition a (parametric) interval.

In [DLP16], we introduced pIMCs where parametric interval endpoints are limited to linear combination of parameters. In this chapter, we extend the pIMC model by allowing rational functions over parameters as endpoints of parametric intervals. Given a pIMC $\mathcal{P} = (S, s_0, P, L, \mathbb{X})$ and a valuation v, we write $v(\mathcal{P})$ for the IMC (S, s_0, P_v, L) obtained by replacing the transition function P from $\mathcal{P}$ with the function $P_v\colon S \times S \to \mathbb{I}$ defined by $P_v(s, s') = v(P(s, s'))$ for all $s, s' \in S$. The IMC $v(\mathcal{P})$ is called an *instance* of pIMC $\mathcal{P}$. Finally, depending on the semantics chosen for IMCs, three satisfaction relations can be defined between MCs and pIMCs. They are written $\models^a_{\texttt{pI}}$, $\models^d_{\texttt{pI}}$, and $\models^o_{\texttt{pI}}$ and defined as follows: $\mathcal{M} \models^a_{\texttt{pI}} \mathcal{P}$ (resp. $\models^d_{\texttt{pI}}$, $\models^o_{\texttt{pI}}$) if and only if there exists at least one IMC I instance of $\mathcal{P}$ such that $\mathcal{M} \models^a_{\texttt{I}} I$ (resp. $\models^d_{\texttt{I}}$, $\models^o_{\texttt{I}}$).

Example 7. Consider the pIMC $\mathcal{P} = (S, s_0, P, L, \mathbb{X})$ given in Figure 1.7. The set of states S and the labelling function are the same as in the MC and the IMC presented in Figures 1.1 and 1.3 respectively. The set of parameters $\mathbb{X}$ has two elements p and q. Finally, the parametric intervals from the transition function P are given by the edge labelling (*e.g.*, $P(s_1, s_3) = [0.3, q]$, $P(s_2, s_4) = [0, 0.5]$, and $P(s_3, s_3) = [1, 1]$). Note that the IMC I from Figure 1.3 is an instance of $\mathcal{P}$ (by assigning the value 0.6 to the parameter p and 0.5 to q). Furthermore, as said in Example 6, the Markov chains $\mathcal{M}_1$ and $\mathcal{M}_2$ (from Figures 1.1 and 1.4 respectively) satisfy I w.r.t. $\models^a_{\texttt{I}}$, therefore $\mathcal{M}_1$ and $\mathcal{M}_2$ satisfy $\mathcal{P}$ w.r.t. $\models^a_{\texttt{pI}}$.

2. Indeed, when $0 \leq v(f_1) \leq v(f_2) \leq 1$ is not respected, the interval is inconsistent and therefore empty.

In the following, we consider that the size of a pMC, IMC, or pIMC $\mathcal{L}$, written $|\mathcal{L}|$, corresponds to its number of states plus its number of transitions not reduced to 0, $[0,0]$ or $\varnothing$. We will also often need to consider the predecessors (`Pred`), and the successors (`Succ`) of some given states. Given a pIMC with a set of states S, a state s in S, and a subset S' of S, we write:

— $\texttt{Pred}(s) = \{\, s' \in S \mid P(s',s) \notin \{\, \varnothing, [0,0] \,\} \,\}$

— $\texttt{Succ}(s) = \{\, s' \in S \mid P(s,s') \notin \{\, \varnothing, [0,0] \,\} \,\}$

— $\texttt{Pred}(S') = \bigcup_{s' \in S'} \texttt{Pred}(s')$

— $\texttt{Succ}(S') = \bigcup_{s' \in S'} \texttt{Succ}(s')$

1.3.2 Abstraction model comparisons

IMC, pMC, and pIMC are three Markov chain abstraction models. In order to compare their expressiveness and compactness, we introduce the comparison operators $\sqsubseteq$ and $\equiv$. Let $(\mathtt{L}_1, \models_1)$ and $(\mathtt{L}_2, \models_2)$ be two Markov chain abstraction models containing respectively $\mathcal{L}_1$ and $\mathcal{L}_2$. We say that $\mathcal{L}_1$ is entailed by $\mathcal{L}_2$, written $\mathcal{L}_1 \sqsubseteq \mathcal{L}_2$, if and only if all MCs satisfying $\mathcal{L}_1$ satisfy $\mathcal{L}_2$ modulo bisimilarity. (*i.e.*, $\forall \mathcal{M} \models_1 \mathcal{L}_1, \exists \mathcal{M}' \models_2 \mathcal{L}_2$ such that $\mathcal{M}$ is bisimilar to $\mathcal{M}'$). We say that $\mathcal{L}_1$ is (semantically) equivalent to $\mathcal{L}_2$, written $\mathcal{L}_1 \equiv \mathcal{L}_2$, if and only if $\mathcal{L}_1 \sqsubseteq \mathcal{L}_2$ and $\mathcal{L}_2 \sqsubseteq \mathcal{L}_1$. Definition 11 introduces succinctness based on the sizes of the abstractions.

Definition 11 (Succinctness)**.** Let $(\mathtt{L}_1, \models_1)$ and $(\mathtt{L}_2, \models_2)$ be two Markov chain abstraction models. $\mathtt{L}_1$ is at least as succinct as $\mathtt{L}_2$, written $\mathtt{L}_1 \leq \mathtt{L}_2$, if and only if there exists a polynomial p such that for every $\mathcal{L}_2 \in \mathtt{L}_2$, there exists $\mathcal{L}_1 \in \mathtt{L}_1$ such that $\mathcal{L}_1 \equiv \mathcal{L}_2$ and $|\mathcal{L}_1| \leq p(|\mathcal{L}_2|)$. Moreover, $\mathtt{L}_1$ is strictly more succinct than $\mathtt{L}_2$, written $\mathtt{L}_1 < \mathtt{L}_2$, if and only if $\mathtt{L}_1 \leq \mathtt{L}_2$ and $\mathtt{L}_2 \not\leq \mathtt{L}_1$.

We start with a comparison of the succinctness of the pMC and IMC abstractions. Since pMCs allow the expression of dependencies between the probabilities assigned to distinct transitions while IMCs allow all transitions to be independent, it is clear that there are pMCs without any equivalent IMCs (regardless of the IMC semantics used), therefore $(\texttt{IMC}, \models_{\texttt{I}}^{\texttt{o}}) \not\leq (\texttt{pMC}, \models_{\texttt{p}})$, $(\texttt{IMC}, \models_{\texttt{I}}^{\texttt{d}}) \not\leq (\texttt{pMC}, \models_{\texttt{p}})$, and $(\texttt{IMC}, \models_{\texttt{I}}^{\texttt{a}}) \not\leq (\texttt{pMC}, \models_{\texttt{p}})$. On the other hand, IMCs with the IMDP and at-every-step semantics allow unbounded unfolding of the state space with different probabilistic choices, which implies that pMCs would need infinitely many states to represent the same set of MC implementations. Finally, we show that the set of MC implementations of an IMC with the once-and-for-all semantics can also be represented with a pMC of the same size as the original IMC. As a consequence, pMCs are strictly more succinct than IMCs with the once-and-for-all semantics. This is formalized in the following lemma.

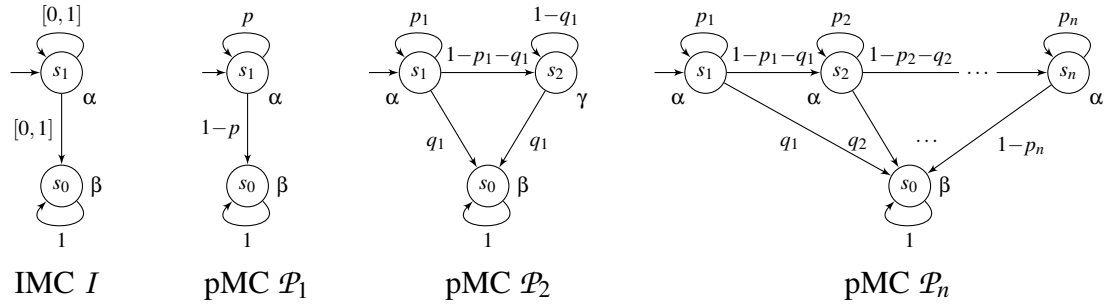


Figure 1.8 – IMC I with three pMCs $\mathcal{P}_1$, $\mathcal{P}_2$, and $\mathcal{P}_n$.

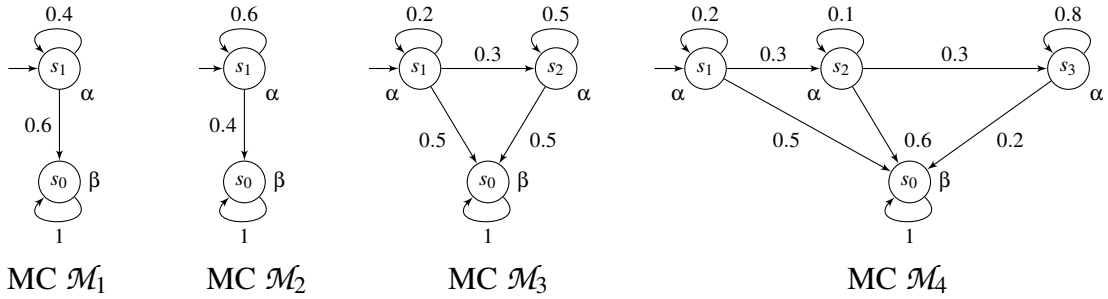


Figure 1.9 – MCs satisfying the IMC and/or one of the pMCs of Figure 1.8.

Lemma 1. pMC *and* IMC *abstraction models are mostly not comparable in terms of succinctness: (1)* $(\texttt{pMC}, \models_{\texttt{p}}) \not\leq (\texttt{IMC}, \models^{\texttt{a}}_{\texttt{I}})$*, (2)* $(\texttt{pMC}, \models_{\texttt{p}}) \not\leq (\texttt{IMC}, \models^{\texttt{d}}_{\texttt{I}})$*, (3)* $(\texttt{IMC}, \models^{\texttt{a}}_{\texttt{I}}) \not\leq (\texttt{pMC}, \models_{\texttt{p}})$*, and (4)* $(\texttt{IMC}, \models^{\texttt{d}}_{\texttt{I}}) \not\leq (\texttt{pMC}, \models_{\texttt{p}})$*. However,* pMC*s are stricly more succinct than* IMC*s with the once-and-for-all semantics: (5)* $(\texttt{pMC}, \models_{\texttt{p}}) < (\texttt{IMC}, \models^{\texttt{o}}_{\texttt{I}})$.

This result is illustrated in Figures 1.8 and 1.9. For the sake of conciseness, we only give some a proof-sketch. A complete proof is provided in [BDF+18]. Let $(\texttt{L}_1, \models_1)$ and $(\texttt{L}_2, \models_2)$ be two Markov chain abstraction models. Recall that according to the succinctness definition (cf. Definition 11) $\texttt{L}_1 \not\leq \texttt{L}_2$ if there exists $\mathcal{L}_2 \in \texttt{L}_2$ such that either $\mathcal{L}_1 \not\equiv \mathcal{L}_2$ for all $\mathcal{L}_1 \in \texttt{L}_1$, or the only way to have $\mathcal{L}_1 \equiv \mathcal{L}_2$ is with $\mathcal{L}_1$ exponentially larger than $\mathcal{L}_2$.

(1-2) It is easy to see that all of the MCs given in Figure 1.9 satisfy I, given in Figure 1.8 with $\models^a_{\texttt{I}}$ and $\models^d_{\texttt{I}}$. Because State s_1 can be "unfolded" as many times as we want, with different output probabilities, we would need pMCs such as $\mathcal{P}_n$ from Figure 1.8 with infinitely many states to represent the same set of MC implementations.

(3-4) pMC $\mathcal{P}_2$ from Figure 1.8 enforces the transitions from s_1 to s_0 and the transition from s_2 to s_0 to have the same probability. Since IMCs do not allow such dependencies between transitions, the set of MCs satisfying $\mathcal{P}_2$ cannot be matched by any IMC semantics.

(5) For the same argument as above, we have $(\texttt{IMC}, \models^{\texttt{o}}_{\texttt{I}}) \not\leq (\texttt{pMC}, \models_{\texttt{p}})$. On the other hand, any transition equipped with an interval of the form $[l,u]$ (IMC), with $0 \leq l \leq u \leq 1$ can be matched with a transition with probability $l + (u-l) \cdot p$ (pMC), where p is a fresh parameter. Since this transformation preserves the size of the original IMC, we have $(\texttt{pMC}, \models_{\texttt{p}}) < \texttt{IMC}, \models^{\texttt{o}}_{\texttt{I}}$.

We now compare pMCs and IMCs to pIMCs. We show that $(\texttt{pIMC}, \models^{o}_{\texttt{pI}})$ is equivalent to $(\texttt{pMC}, \models_{\texttt{p}})$ and pIMC is strictly more succinct than IMC for the three semantics.
Our comparison results are presented in the following proposition.

Proposition 2. *The Markov chain abstraction models can be ordered as follows w.r.t. succinctness: (1)* $(\texttt{pIMC}, \models^{o}_{\texttt{pI}}) < (\texttt{IMC}, \models^{\texttt{o}}_{\texttt{I}})$*, (2)* $(\texttt{pIMC}, \models^{d}_{\texttt{pI}}) < (\texttt{IMC}, \models^{\texttt{d}}_{\texttt{I}})$*, (3)* $(\texttt{pIMC}, \models^{a}_{\texttt{pI}}) < (\texttt{IMC}, \models^{\texttt{a}}_{\texttt{I}})$*, (4)* $(\texttt{pIMC}, \models^{d}_{\texttt{pI}}) < (\texttt{pMC}, \models_{\texttt{p}})$*, (5)* $(\texttt{pIMC}, \models^{a}_{\texttt{pI}}) < (\texttt{pMC}, \models_{\texttt{p}})$*, and (6)* $(\texttt{pIMC}, \models^{o}_{\texttt{pI}}) \equiv (\texttt{pMC}, \models_{\texttt{p}})$.

The proofs of items (1-5) are straightforward. Item 6 is more involved but can be obtained using a construction that derives, from any given pIMC $\mathcal{P}$, a pMC I that represents the same set of MC implementations up to bisimilarity. The intuition of this construction is to use fresh parameters and several copies of the transitions in $\mathcal{P}$ to encode the constraints on the interval endpoints. Details of this construction are provided in [BDF+18].

1.4 Qualitative properties

As seen above, pIMCs are a succinct abstraction formalism for MCs. The aim of this section is to investigate qualitative properties for pIMCs, *i.e.*, properties that can be evaluated at the specification (pIMC) level, but that entail properties on its MC implementations. pIMC specifications are very expressive as they allow the abstraction of transition probabilities using both intervals and parameters. Unfortunately, as it is the case for IMCs, this allows the expression of incorrect specifications. In the IMC setting, this is the case either when some intervals are ill-formed or when there is no probability distribution matching the interval constraints of the outgoing transitions of some reachable state. In this case, no MC implementation exists that satisfies the IMC specification. Deciding whether an implementation that satisfies a given specification exists is called the *consistency* problem. In the pIMC setting, the consistency problem is made more complex because of the parameters which can also induce inconsistencies in some cases. One could also be interested in verifying whether there exists an implementation that reaches some target states/labels, and if so, propose a parameter valuation ensuring this property. This problem is called the consistent reachability problem.

1.4.1 Existential consistency

A pIMC $\mathcal{P}$ is existentially consistent if and only if there exists a MC $\mathcal{M}$ satisfying $\mathcal{P}$ (*i.e.*, there exists a MC $\mathcal{M}$ satisfying an IMC I instance of $\mathcal{P}$). As seen in Section 1.3, pIMCs are equipped with three semantics: once-and-for-all ($\models^o_{\mathtt{pI}}$), IMDP ($\models^d_{\mathtt{pI}}$), and at-every-step ($\models^a_{\mathtt{pI}}$). Recall that $\models^o_{\mathtt{pI}}$ imposes that implementations need to have the same graph structure as (or a substructure of) the corresponding specification, up to renaming. In contrast, $\models^d_{\mathtt{pI}}$ and $\models^a_{\mathtt{pI}}$ allow implementations to have a different graph structure. It therefore seems that some pIMCs could be inconsistent w.r.t $\models^o_{\mathtt{pI}}$ while being consistent w.r.t $\models^a_{\mathtt{pI}}$. On the other hand, checking the consistency w.r.t $\models^o_{\mathtt{pI}}$ seems easier because of the fixed graph structure.

In [Del15], I proved that $\models^a_{\mathtt{pI}}$ and $\models^o_{\mathtt{pI}}$ semantics are equivalent w.r.t. existential consistency. Together with Proposition 1, this result ensures that the three semantics $\models^d_{\mathtt{pI}}$, $\models^a_{\mathtt{pI}}$, and $\models^o_{\mathtt{pI}}$ are equivalent w.r.t. existential consistency. Based on this result of semantics equivalence we propose a CSP encoding, written $\mathbf{C}_{\exists\mathbf{c}}$, for verifying the existential consistency property for pIMCs.

Let $\mathcal{P} = (S,s_0,P,L,\mathbb{X})$ be a pIMC, we write $\mathbf{C}_{\exists\mathbf{c}}(\mathcal{P})$ for the CSP produced by $\mathbf{C}_{\exists\mathbf{c}}$ according to $\mathcal{P}$. Any solution of $\mathbf{C}_{\exists\mathbf{c}}(\mathcal{P})$ will correspond to a MC satisfying $\mathcal{P}$. In $\mathbf{C}_{\exists\mathbf{c}}(\mathcal{P}) = (X_\mathbf{c}, D_\mathbf{c}, C_\mathbf{c})$, the sets of variables $X_\mathbf{c}$ and domains $D_\mathbf{c}$ are as follows: we use one variable π_p with domain $[0,1]$ per parameter p in $\mathbb{X}$; one variable $\theta^{s'}_s$ with domain $[0,1]$ per transition (s,s') in $\{\{s\} \times \mathtt{Succ}(s) \mid s \in S\}$; and one Boolean variable ρ_s per state s in S. These Boolean variables will indicate for each state whether it appears in the MC solution of the CSP (*i.e.*, in the MC satisfying the pIMC $\mathcal{P}$). Finally, we use the following constraints, for each state $s \in S$:

(1) ρ_{s_0}, (only for s_0)

(2) $\neg\rho_s \Leftrightarrow \Sigma_{s' \in \mathtt{Pred}(s)\setminus\{s\}} \theta^s_{s'} = 0$, (only for $s \neq s_0$)

(3) $\neg\rho_s \Leftrightarrow \Sigma_{s' \in \mathtt{Succ}(s)} \theta^{s'}_s = 0$

(4) $\rho_s \Leftrightarrow \Sigma_{s' \in \mathtt{Succ}(s)} \theta^{s'}_s = 1$

(5) $\rho_s \Rightarrow \theta^{s'}_s \in P(s,s')$, for all $s' \in \mathtt{Succ}(s)$

Recall that given a pIMC $\mathcal{P}$ the objective of the CSP $\mathbf{C}_{\exists\mathbf{c}}(\mathcal{P})$ is to construct a MC $\mathcal{M}$ satisfying $\mathcal{P}$. Constraint **(1)** states that the initial state s_0 appears in $\mathcal{M}$. Constraint **(2)** ensures that for each non-initial state s, variable ρ_s is set to false if and only if s is not reachable from its predecessors. Constraint **(4)** ensures that if a state s appears in $\mathcal{M}$, then its outgoing transitions form a probability distribution. On the contrary, constraint **(3)** propagates non-appearing states (*i.e.*, if a state s does not appear in $\mathcal{M}$ then all its outgoing transitions are set to zero). Finally, constraint **(5)** states that, for all appearing states, the outgoing transition probabilities must be selected inside the specified intervals.

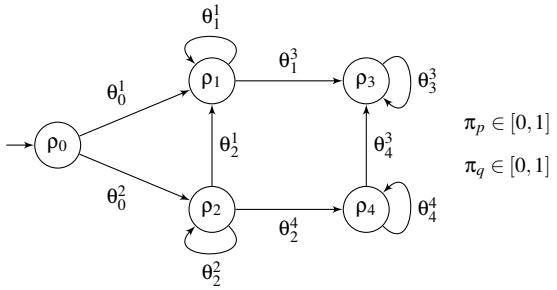


Figure 1.10 – Variables in the CSP produced by $\mathbf{C}_{\exists\mathbf{c}}$ for the pIMC $\mathcal{P}$ from Fig. 1.7

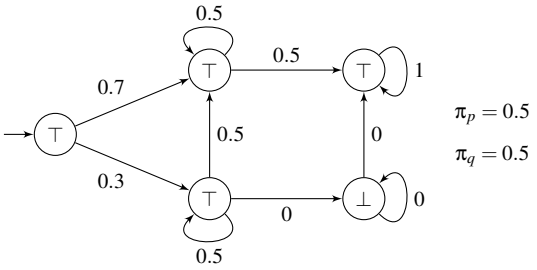


Figure 1.11 – A solution to the CSP $\mathbf{C}_{\exists\mathbf{c}}(\mathcal{P})$ for the pIMC $\mathcal{P}$ from Fig. 1.7

Example 8. Consider the pIMC $\mathcal{P}$ given in Figure 1.7. Figure 1.10 describes the variables in $\mathbf{C}_{\exists\mathbf{c}}(\mathcal{P})$: one variable per transition (*e.g.*, θ_0^1, θ_0^2, θ_1^1,...), one Boolean variable per state (*e.g.*, ρ_0, ρ_1,...), and one variable per parameter (π_p and π_q). The following constraints correspond to the constraints **(2)**, **(3)**, **(4)**, and **(5)** generated by our encoding $\mathbf{C}_{\exists\mathbf{c}}$ for the state 2 of $\mathcal{P}$:

$$\neg\rho_2 \Leftrightarrow \theta_0^2 = 0 \qquad \rho_2 \Rightarrow 0 \leq \theta_2^1 \leq \pi_p$$

$$\neg\rho_2 \Leftrightarrow \theta_2^1 + \theta_2^2 + \theta_2^4 = 0 \qquad \rho_2 \Rightarrow 0.2 \leq \theta_2^2 \leq \pi_p$$

$$\rho_2 \Leftrightarrow \theta_2^1 + \theta_2^2 + \theta_2^4 = 1 \qquad \rho_2 \Rightarrow 0 \leq \theta_2^4 \leq 0.5$$

Finally, Figure 1.11 describes a solution for the CSP $\mathbf{C}_{\exists\mathbf{c}}(\mathcal{P})$. Note that given a solution of a pIMC encoded by $\mathbf{C}_{\exists\mathbf{c}}$, one can construct a MC satisfying the given pIMC w.r.t. $\models_{\mathtt{I}}^{o}$ by keeping all states s such that ρ_s is equal to true and considering the transition function given by the probabilities in the $\theta_s^{s'}$ variables. The following proposition shows that our encoding works as expected. The proof is provided in [BDF+18].

Proposition 3. *A* pIMC $\mathcal{P}$ *is existentially consistent if and only if* $\mathbf{C}_{\exists\mathbf{c}}(\mathcal{P})$ *is satisfiable.*

We note that this existential consistency encoding is linear in the size of the pIMC, which is better than the exponential encoding we proposed in [DLP16], which enumerates the powerset of the states in the pIMC resulting in deep nesting of conjunctions and disjunctions.

1.4.2 Qualitative reachability

Let $\mathcal{P} = (S, s_0, P, L, \mathbb{X})$ be a pIMC and $\Gamma \subseteq A$ be a state label. We say that Γ is *existentially reachable* in $\mathcal{P}$ if and only if there exists an implementation $\mathcal{M}$ of $\mathcal{P}$ where Γ is reachable (*i.e.*, $\mathbb{P}_{\mathcal{M}}(\Diamond\Gamma) > 0$). In a dual way, we say that Γ is *universally reachable* in $\mathcal{P}$ if and only if Γ is reachable in any implementation $\mathcal{M}$ of $\mathcal{P}$. As for existential consistency, we use a result from [Del15] that states that the $\models_{\mathtt{I}}^{a}$ and the $\models_{\mathtt{I}}^{o}$ pIMC semantics are equivalent w.r.t. existential

(and universal) reachability. As for the consistency problem, we get by Proposition 1 that the three IMC semantics are equivalent w.r.t. existential (and universal) reachability. Note first that in our $\mathbf{C}_{\exists\mathbf{c}}$ encoding each ρ_s variable indicates if the state s appears in the constructed Markov chain. However, the ρ_s variable does not indicate if the state s is reachable from the initial state, but only if it is reachable from at least one other state (*i.e.*, possibly different from s_0). Indeed, if the graph representation of the constructed Markov chain has strongly connected components (SCCs for short), then all ρ_s variables in one SCC may be set to true while this SCC may be unreachable from the initial state. This is not an issue in the case of the consistency problem. Indeed, if a Markov chain containing an unreachable SCC is proved consistent, then it is also consistent without this unreachable SCC. However, in the case of the reachability problem, these SCCs are an issue. The following encoding therefore takes into account these isolated SCCs such that ρ_s variables are set to true if and only if they are all reachable from the initial state. This encoding will solve the qualitative reachability problems (*i.e.*, checking qualitative reachability from the initial state). We propose a new CSP encoding, written $\mathbf{C}_{\exists\mathbf{r}}$, that extends $\mathbf{C}_{\exists\mathbf{c}}$, for verifying these properties. Formally, CSP $\mathbf{C}_{\exists\mathbf{r}}(\mathcal{P}) = (X_\mathbf{c} \cup X', D_\mathbf{c} \cup D', C_\mathbf{c} \cup C')$ is such that $(X_\mathbf{c}, D_\mathbf{c}, C_\mathbf{c}) = \mathbf{C}_{\exists\mathbf{c}}(\mathcal{P})$, X' contains one integer variable ω_s with domain $[0, |S|]$ per state s in S, D' contains the domains of these variables, and C' is composed of the following constraints for each state $s \in S$:

(6) $\omega_{s_0} = 1$, (only for s_0)

(7) $\omega_s \neq 1$, (only for $s \neq s_0$)

(8) $\rho_s \Leftrightarrow (\omega_s \neq 0)$

(9) $\omega_s > 1 \Rightarrow \bigvee_{s' \in \texttt{Pred}(s) \setminus \{s\}} (\omega_s = \omega_{s'} + 1) \wedge (\theta^s_{s'} > 0)$, (only for $s \neq s_0$)

(10) $\omega_s = 0 \Leftrightarrow \bigwedge_{s' \in \texttt{Pred}(s) \setminus \{s\}} (\omega_{s'} = 0) \vee (\theta^s_{s'} = 0)$, (only for $s \neq s_0$)

Recall first that CSP $\mathbf{C}_{\exists\mathbf{c}}(P)$ constructs a Markov chain $\mathcal{M}$ satisfying $\mathcal{P}$ w.r.t. $\models^o_{\mathrm{I}}$. Informally, for each state s in $\mathcal{M}$ the constraints **(6)**, **(7)**, **(9)**, and **(10)** in $\mathbf{C}_{\exists\mathbf{r}}$ ensure that $\omega_s = k$ if and only if there exists in $\mathcal{M}$ a run from the initial state to s of length $k-1$ with non zero probability; and state s is not reachable in $\mathcal{M}$ from the initial state s_0 if and only if ω_s equals to 0. Finally, constraint **(8)** enforces the Boolean reachability indicator variable ρ_s to be set to true if and only if there exists a run with non zero probability in $\mathcal{M}$ from the initial state s_0 to s (*i.e.*, $\omega_s \neq 0$).
Let S_Γ be the set of states from $\mathcal{P}$ labeled with Γ. Recall that $\mathbf{C}_{\exists\mathbf{r}}(\mathcal{P})$ produces a Markov chain satisfying $\mathcal{P}$ where reachable states s are such that $\rho_s = \mathsf{true}$. As a consequence, Γ is existentially reachable in $\mathcal{P}$ if and only if $\mathbf{C}_{\exists\mathbf{r}}(\mathcal{P})$ admits a solution such that $\bigvee_{s \in S_\Gamma} \rho_s$; and Γ is universally

reachable in $\mathcal{P}$ if and only if $\mathbf{C}_{\exists\mathbf{r}}(\mathcal{P})$ admits no solution such that $\bigwedge_{s\in S_\Gamma}\neg\rho_s$. This is formalised in the following proposition, whose proof is provided in [BDF+18].

Proposition 4. *Let $\mathcal{P}=(S,s_0,P,L,\mathbb{X})$ be a* pIMC*, $\Gamma\subseteq A$ be a state label, $S_\Gamma=\{s\mid L(s)=\Gamma\}$, and $(X_\mathbf{r},D_\mathbf{r},C_\mathbf{r})$ be the* CSP *$\mathbf{C}_{\exists\mathbf{r}}(\mathcal{P})$.*

- CSP $(X_\mathbf{r},D_\mathbf{r},C_\mathbf{r}\cup\bigvee_{s\in S_\Gamma}\rho_s)$ *is satisfiable if and only if* Γ *is existentially reachable in* $\mathcal{P}$
- CSP $(X_\mathbf{r},D_\mathbf{r},C_\mathbf{r}\cup\bigwedge_{s\in S_\Gamma}\neg\rho_s)$ *is unsatisfiable if and only if* Γ *is universally reachable in* $\mathcal{P}$

As for the existential consistency problem, we have an exponential gain in terms of size of the encoding compared to the one we proposed in [DLP16]: the number of constraints and variables in $\mathbf{C}_{\exists\mathbf{r}}$ is linear in terms of the size of the encoded pIMC.

Remark. In $\mathbf{C}_{\exists\mathbf{r}}$ constraints **(2)** inherited from $\mathbf{C}_{\exists\mathbf{c}}$ are entailed by constraints **(8)** and **(10)** added to $\mathbf{C}_{\exists\mathbf{r}}$. Thus, in a practical approach one may ignore constraints **(2)** from $\mathbf{C}_{\exists\mathbf{c}}$ if they do not improve the solver performance.

1.5 Quantitative properties

We now move to the verification of quantitative reachability properties in pIMCs. Quantitative reachability has already been investigated in the context of pMCs and IMCs with the once-and-for-all semantics. In this section, we propose a major theoretical contribution: a theorem showing that the three IMC semantics are equivalent with respect to quantitative reachability, which allows the extension of all results from [WTO+11, BLW13] to the at-every-step semantics. Based on this result, we also extend the CSP encodings introduced in Section 1.4 in order to solve quantitative reachability properties on pIMCs regardless of their semantics.

1.5.1 Equivalence of all semantics w.r.t quantitative reachability

Given an IMC $I=(S,s_0,P,L)$ and a state label $\Gamma\subseteq A$, a quantitative reachability property on I is a property of the type $\mathbb{P}_I(\Diamond\Gamma)\sim p$, where $0\leq p\leq 1$ and $\sim\in\{\leq,<,>,\geq\}$. Such a property is verified if and only if there exists a MC $\mathcal{M}$ satisfying I (with the chosen semantics) such that $\mathbb{P}_\mathcal{M}(\Diamond\Gamma)\sim p$. While the techniques we propose here work for all values of p, the techniques for *qualitative* reachability properties are usually more efficient when $p=0$ or 1.
As explained above, existing techniques and tools for verifying quantitative reachability properties on IMCs only focus on the once-and-for-all and the IMDP semantics. However, to the

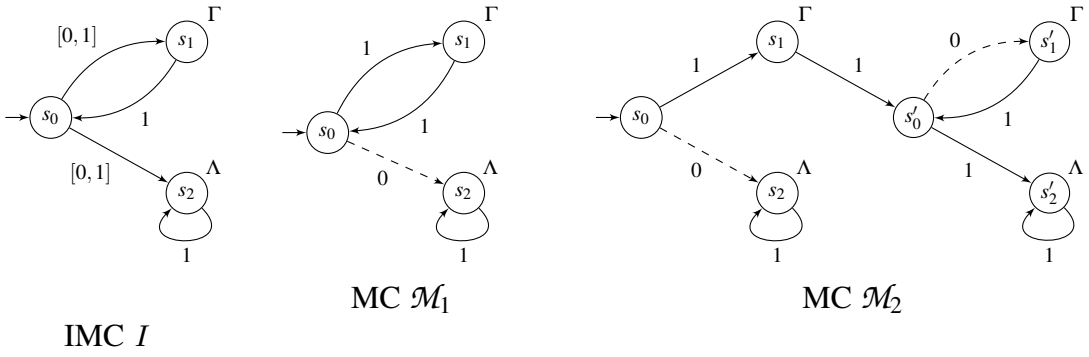


Figure 1.12 – IMC I, MC $\mathcal{M}_1$, and MC $\mathcal{M}_2$ such that property $\mathbb{P}_{=1}(\mathrm{X}\,(\Gamma \wedge \mathbb{P}_{=1}(\Diamond\Lambda)))$ holds for $\mathcal{M}_2$, does not hold for $\mathcal{M}_1$, holds for I w.r.t. $\models^a_{\mathtt{I}}$, and does not hold for I w.r.t. $\models^o_{\mathtt{I}}$.

best of our knowledge, there are no works addressing the same problem with the at-every-step semantics or showing that addressing the problem in the once-and-for-all and IMDP setting is sufficiently general. The following theorem fills this theoretical gap by proving that the three IMC semantics are equivalent w.r.t quantitative reachability. In other words, for all MCs $\mathcal{M}$ such that $\mathcal{M} \models^a_{\mathtt{I}} I$ or $\mathcal{M} \models^d_{\mathtt{I}} I$ and for all state labels Γ, there exist MCs $\mathcal{M}_{\leq}$ and $\mathcal{M}_{\geq}$ such that $\mathcal{M}_{\leq} \models^o_{\mathtt{I}} I$, $\mathcal{M}_{\geq} \models^o_{\mathtt{I}} I$, and $\mathbb{P}_{\mathcal{M}_{\leq}}(\Diamond\Gamma) \leq \mathbb{P}_{\mathcal{M}}(\Diamond\Gamma) \leq \mathbb{P}_{\mathcal{M}_{\geq}}(\Diamond\Gamma)$. Informally, MC implementations for the at-every-step semantics may contain several copies of the original IMC states, each with a different probability of eventually reaching Γ. We therefore build $\mathcal{M}_{\leq}$ (resp. $\mathcal{M}_{\geq}$) by choosing as sole representative of a given IMC state the one from $\mathcal{M}$ that has the lowest (resp. highest) probability of eventually reaching Γ. This way we ensure that $\mathcal{M}_{\leq}$ (resp. $\mathcal{M}_{\geq}$) satisfies the original IMC with the once-and-for-all semantics and has a lower (resp. higher) probability of eventually reaching Γ than $\mathcal{M}$. This is formalised in the following theorem. The proof, which is particularly intricate, can be found in [BDF+18].

Theorem 1. *Let $I = (S, s_0, P, L)$ be an* IMC*, $\Gamma \subseteq A$ be a state label, $\sim \in \{\leq, <, >, \geq\}$, and $0 < p < 1$. I satisfies $\mathbb{P}_I(\Diamond\Gamma) \sim p$ with the once-and-for-all semantics if and only if I satisfies $\mathbb{P}_I(\Diamond\Gamma) \sim p$ with the IMDP semantics if and only if I satisfies $\mathbb{P}_I(\Diamond\Gamma) \sim p$ with the at-every-step semantics.*

Thus, the three semantics $\models^o_{\mathtt{I}}$, $\models^d_{\mathtt{I}}$, and $\models^a_{\mathtt{I}}$ are equivalent with respect to the minimal and the maximal quantitative reachability properties. However, note that the equivalence for reachability properties does not trivially extend to more general properties. For instance, consider the formula $\mathbb{P}_{=1}(\mathrm{X}\,(\Gamma \wedge \mathbb{P}_{=1}(\Diamond\Lambda)))$, where Γ and Λ are two state labels. This property states that the probability of visiting a state labeled with Γ in the next step and almost certainly reaching a state labeled with Λ afterwards is 1. Figure 1.12 contains an IMC I such that this property does

not hold w.r.t. $\models^o_{\mathtt{I}}$ while it holds w.r.t. $\models^a_{\mathtt{I}}$. Indeed, under the $\models^o_{\mathtt{I}}$ semantics, enforcing that the probability of visiting a state labeled with Γ as the next state is 1 imposes that the branch leading to Λ is given a probability 0, hence preventing from almost certainly visiting a state labeled with Λ afterwards (cf. the MC $\mathcal{M}_1$ in Figure 1.12). On the contrary, the $\models^a_{\mathtt{I}}$ semantics allows one to unfold the structure, therefore to first assign a probability 1 to the transition leading to Γ and then give a probability 1 to the transition leading to Λ (cf. the MC $\mathcal{M}_2$ in Figure 1.12). Note that the same would be possible under the $\models^d_{\mathtt{I}}$ semantics.

1.5.2 Constraint encodings

Note that the result from Theorem 1 naturally extends to pIMCs. We therefore exploit this result to construct a CSP encoding for verifying quantitative reachability properties in pIMCs. As in Section 1.4, we extend the CSP $\mathbf{C}_{\exists\mathbf{c}}$, that produces a correct MC implementation for the given pIMC, by imposing that this MC implementation satisfies the given quantitative reachability property. In order to compute the probability of reaching state label Γ at the MC level, we use standard techniques from [BK08] that require the partitioning of the state space into three sets $S_{=1}$, $S_{=0}$, and $S_?$ that correspond to a subset of states reaching Γ with probability 1, a subset of states from which Γ cannot be reached, and the remaining states, respectively. Once this partition is chosen, the reachability probabilities of all states in $S_?$ are computed as the unique solution of a linear equation system (see [BK08], Theorem 10.19, p.766). We now explain how we identify states from $S_{=0}, S_{=1}$ and $S_?$, and how we encode the linear equation system, which leads to the resolution of quantitative reachability.

Let $\mathcal{P} = (S, s_0, P, L, \mathbb{X})$ be a pIMC and $\Gamma \subseteq A$ be a state label. We start by setting $S_{=1} = \{ s \mid L(s) = \Gamma \}$. We then extend $\mathbf{C}_{\exists\mathbf{r}}(\mathcal{P})$ in order to identify the set $S_{=0}$. Let $\mathbf{C}'_{\exists\mathbf{r}}(\mathcal{P}, \Gamma) = (X_\mathbf{r} \cup X', D_\mathbf{r} \cup D', C_\mathbf{r} \cup C')$ be such that $(X_\mathbf{r}, D_\mathbf{r}, C_\mathbf{r}) = \mathbf{C}_{\exists\mathbf{r}}(\mathcal{P})$, X' contains one Boolean variable λ_s and one integer variable α_s with domain $[0, |S|]$ per state s in S, D' contains the domains of these variables, and C' is composed of the following constraints for each state $s \in S$:

(11) $\alpha_s = 1$, if $\Gamma = L(s)$

(12) $\alpha_s \neq 1$, if $\Gamma \neq L(s)$

(13) $\lambda_s \Leftrightarrow (\rho_s \wedge (\alpha_s \neq 0))$

(14) $\alpha_s > 1 \Rightarrow \bigvee_{s' \in \mathtt{Succ}(s) \setminus \{s\}} (\alpha_s = \alpha_{s'} + 1) \wedge (\theta_s^{s'} > 0)$, if $\Gamma \neq L(s)$

(15) $\alpha_s = 0 \Leftrightarrow \bigwedge_{s' \in \mathtt{Succ}(s) \setminus \{s\}} (\alpha_{s'} = 0) \vee (\theta_s^{s'} = 0)$, if $\Gamma \neq L(s)$

Note that variables α_s play a symmetric role to variables ω_s from $\mathbf{C}_{\exists\mathbf{r}}$: instead of indicating the

existence of a run from s_0 to s, they characterize the existence of a run from s to a state labeled with Γ. In addition, due to constraint **(13)**, variables λ_s are set to true if and only if there exists a run with non zero probability from the initial state s_0 to a state labeled with Γ visiting s. Thus, Γ cannot be reached from states such that $\lambda_s = \mathsf{false}$. Therefore, $S_{=0} = \{ s \mid \lambda_s = \mathsf{false} \}$, which is formalised in Proposition 5.

Proposition 5. *Let $\mathcal{P} = (S, s_0, P, L, \mathbb{X})$ be a* pIMC *and $\Gamma \subseteq A$ be a state label. There exists a* MC *$\mathcal{M} \models^a_{\mathtt{pI}} \mathcal{P}$ if and only if there exists a valuation v solution of the* CSP $\mathbf{C}'_{\exists \mathbf{r}}(\mathcal{P}, \Gamma)$ *such that for each state $s \in S$: $v(\lambda_s)$ is equal to* true *if and only if $\mathbb{P}^s_{\mathcal{M}}(\Diamond \Gamma) \neq 0$ and s is reachable from s_0.*

Finally, we encode the equation system from [BK08] in a last CSP encoding that extends $\mathbf{C}'_{\exists \mathbf{r}}$. Let $\mathbf{C}_{\exists \bar{\mathbf{r}}}(\mathcal{P}, \Gamma) = (X'_{\mathbf{r}} \cup X', D'_{\mathbf{r}} \cup D', C'_{\mathbf{r}} \cup C')$ be such that $(X'_{\mathbf{r}}, D'_{\mathbf{r}}, C'_{\mathbf{r}}) = \mathbf{C}'_{\exists \mathbf{r}}(\mathcal{P}, \Gamma)$, X' contains one variable γ_s with domain $[0,1]$ per state s in S, D' contains the domains of these variables, and C' is composed of the following constraints for each state $s \in S$:

$$\textbf{(16)}\ \ \neg\lambda_s \Rightarrow \gamma_s = 0$$

$$\textbf{(17)}\ \ \lambda_s \Rightarrow \gamma_s = 1, \qquad \text{if } \Gamma = L(s)$$

$$\textbf{(18)}\ \ \lambda_s \Rightarrow \gamma_s = \Sigma_{s' \in \mathtt{Succ}(s)} \gamma_{s'} \theta^{s'}_s, \qquad \text{if } \Gamma \neq L(s)$$

As a consequence, variables γ_s encode the probability with which state s eventually reaches Γ when s is reachable from the initial state and, 0 otherwise.

Proposition 6. *Let $\mathcal{P} = (S, s_0, P, L, \mathbb{X})$ be a* pIMC *and $\Gamma \subseteq A$ be a state label. There exists a* MC *$\mathcal{M} \models^a_{\mathtt{pI}} \mathcal{P}$ if and only if there exists a valuation v solution of the* CSP $\mathbf{C}_{\exists \bar{\mathbf{r}}}(\mathcal{P}, \Gamma)$ *such that $v(\gamma_s)$ is equal to $\mathbb{P}^s_{\mathcal{M}}(\Diamond \Gamma)$ if s is reachable from the initial state s_0 in $\mathcal{M}$, and is equal to 0 otherwise.*

Let $p \in [0,1] \subseteq \mathbb{R}$ be a probability bound. Adding the constraint $\gamma_{s_0} \sim p$ to the previous $\mathbf{C}_{\exists \bar{\mathbf{r}}}$ encoding allows one to determine if there exists a MC $\mathcal{M} \models^a_{\mathtt{pI}} \mathcal{P}$ such that $\mathbb{P}_{\mathcal{M}}(\Diamond \Gamma) \sim p$. Formally, let $\sim \in \{ \leq, <, \geq, > \}$ be a comparison operator, we write $\not\sim$ for its negation (*e.g.*, $\not\leq$ is $>$). This leads to the following theorem.

Theorem 2. *Let $\mathcal{P} = (S, s_0, P, L, \mathbb{X})$ be a* pIMC*, $\Gamma \subseteq A$ be a label, $p \in [0,1]$, $\sim \in \{ \leq, <, \geq, > \}$ be a comparison operator, and $(X_{\bar{\mathbf{r}}}, D_{\bar{\mathbf{r}}}, C_{\bar{\mathbf{r}}})$ be $\mathbf{C}_{\exists \bar{\mathbf{r}}}(\mathcal{P}, \Gamma)$:*

- CSP $(X_{\bar{\mathbf{r}}}, D_{\bar{\mathbf{r}}}, C_{\bar{\mathbf{r}}} \cup (\gamma_{s_0} \sim p))$ *is satisfiable if and only if $\exists \mathcal{M} \models^a_{\mathtt{pI}} \mathcal{P}$ such that $\mathbb{P}_{\mathcal{M}}(\Diamond \Gamma) \sim p$*
- CSP $(X_{\bar{\mathbf{r}}}, D_{\bar{\mathbf{r}}}, C_{\bar{\mathbf{r}}} \cup (\gamma_{s_0} \not\sim p))$ *is unsatisfiable if and only if $\forall \mathcal{M} \models^a_{\mathtt{pI}} \mathcal{P}$: $\mathbb{P}_{\mathcal{M}}(\Diamond \Gamma) \sim p$*

The full proof, which is quite straightforward, is provided in [BDF+18].

Example 9. Due to the number of constraints in $\mathbf{C}_{\exists\bar{\mathbf{r}}}$, we illustrate our approach on a very small pIMC $\mathcal{P}$ given in the top left of Figure 1.13. In this example, we are interested in two properties: "Does there exist an implementation such that $\mathbb{P}(\Diamond\{\alpha\}) \geq 0.5$?" and "Are all implementations such that $\mathbb{P}(\Diamond\{\alpha\}) \leq 0.5$?". These properties are not trivial because the parameter q appears both on the right side of the interval leading to 1 and on the left side of the interval leading to 2. In the bottom of Figure 1.13, we report the constraints obtained in $\mathbf{C}_{\exists\bar{\mathbf{r}}}(\mathcal{P}, \{\alpha\})$. Some of the most trivial constraints have been removed for readability.

We first consider the CSP obtained by adding the constraint ensuring the first quantitative reachability property: $(QR_1) : \gamma_0 \geq 0.5$. Solving this CSP yields a solution that allows to build a MC $\mathcal{M}$, given in the top right of Figure 1.13, satisfying the quantitative reachability property. The valuations obtained by solving the given CSP are such that $\pi_p = 0, \pi_q = 0.5, \theta_0^1 = \theta_0^2 = 0.5$, and, more importantly, $\gamma_0 = 0.5$.

We now consider the CSP obtained by adding the constraint checking the second quantitative reachability property: $(QR_2) : \gamma_0 > 0.5$. In this case, the CSP is unsatisfiable, which proves that all implementations are such that $\mathbb{P}(\Diamond\{\alpha\}) \leq 0.5$. This is not really surprising as the probability of going to 2 is necessarily greater than the probability of going to 1.

1.6 Prototype implementation and experiments

Our results have been implemented in a prototype tool [3] which generates the above CSP encodings, and CSP encodings from [DLP16] as well. In this section, we first present our benchmark, then we evaluate our tool for the qualitative properties, and we conclude with the quantitative properties.

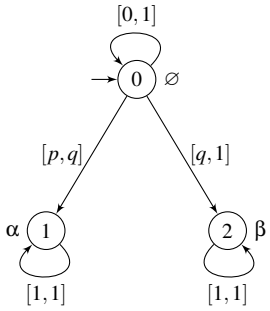
1.6.1 Benchmark

MCs have been used for many decades to model real-life applications. PRISM [KNP11] [4] is a reference for the verification of probabilistic systems. In particular, it is able to verify properties for MCs. As said in Section 1.2, pIMCs correspond to abstractions of MCs. PRISM references several benchmarks based on MCs [5]. Note first that we only consider in this section pIMCs with linear parametric expressions. In this context all CSPs encodings for verifying the

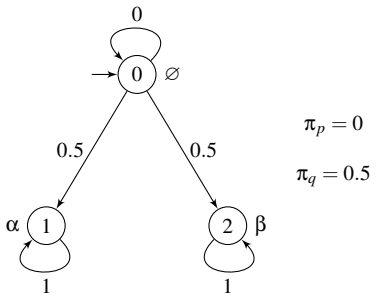
3. All resources, benchmarks, and source code are available online as a Python library at `https://github.com/anicet-bart/pimc_pylib`

4. `http://www.prismmodelchecker.org/`

5. See the category discrete-time Markov chains on the PRISM website



pIMC $\mathcal{P}$



MC $\mathcal{M}$ satisfying $\mathbb{P}(\Diamond\{\alpha\}) \geq 0.5$

(1) ρ_0
(2) $(\neg\rho_1 \Leftrightarrow (\theta_0^1 = 0)) \wedge (\neg\rho_2 \Leftrightarrow (\theta_0^2 = 0))$
(3) $(\neg\rho_1 \Leftrightarrow (\theta_1^1 = 0)) \wedge (\neg\rho_2 \Leftrightarrow (\theta_2^2 = 0))$
(4) $(\rho_0 \Leftrightarrow (\theta_0^0 + \theta_0^1 + \theta_0^2 = 1))$
$\wedge(\rho_1 \Leftrightarrow (\theta_1^1 = 1)) \wedge (\rho_2 \Leftrightarrow (\theta_2^2 = 1))$
(5) $(\rho_0 \Rightarrow (0 \leq \theta_0^0 \leq 1)) \wedge (\rho_0 \Rightarrow (\pi_p \leq \theta_0^1 \leq \pi_q)) \wedge (\rho_0 \Rightarrow (\pi_q \leq \theta_0^2 \leq 1))$
(6) $\omega_0 = 1$
(7) $(\omega_1 \neq 1) \wedge (\omega_2 \neq 1)$
(8) $(\rho_1 \Leftrightarrow \omega_1 \neq 0) \wedge (\rho_2 \Leftrightarrow \omega_2 \neq 0)$
(9) $((\omega_1 > 1) \Rightarrow (\omega_1 = \omega_0 + 1) \wedge \theta_0^1 > 0) \wedge ((\omega_2 > 1) \Rightarrow (\omega_2 = \omega_0 + 1) \wedge \theta_0^2 > 0)$
(10) $((\omega_1 = 0) \Leftrightarrow (\omega_0 = 0) \vee (\theta_0^1 = 0)) \wedge ((\omega_2 = 0) \Leftrightarrow (\omega_0 = 0) \vee (\theta_0^2 = 0))$
(11) $\alpha_1 = 1$
(12) $(\alpha_0 \neq 1) \wedge (\alpha_2 \neq 1)$
(13) $(\lambda_0 \Leftrightarrow (\rho_0 \wedge (\alpha_0 \neq 0))) \wedge (\lambda_1 \Leftrightarrow (\rho_1 \wedge (\alpha_1 \neq 0))) \wedge (\lambda_2 \Leftrightarrow (\rho_2 \wedge (\alpha_2 \neq 0)))$
(14) $((\alpha_0 > 1) \Rightarrow [(\alpha_0 = \alpha_1 + 1) \wedge (\theta_0^1 > 0)] \vee [(\alpha_0 = \alpha_2 + 1) \wedge (\theta_0^2 > 0)])$
$\wedge((\alpha_2 > 1) \Rightarrow [(\alpha_2 = \alpha_2 + 1) \wedge (\theta_2^2 > 0)])$
(15) $(\alpha_0 = 0) \Leftrightarrow [(\alpha_1 = 0) \vee (\theta_0^1 = 0)] \wedge [(\alpha_2 = 0) \vee (\theta_0^2 = 0)]$
(16) $(\neg\lambda_0 \Rightarrow (\gamma_0 = 0)) \wedge (\neg\lambda_1 \Rightarrow (\gamma_1 = 0)) \wedge (\neg\lambda_2 \Rightarrow (\gamma_2 = 0))$
(17) $\lambda_1 \Rightarrow (\gamma_1 = 1)$
(18) $\lambda_0 \Rightarrow (\gamma_0 = \gamma_1 \cdot \theta_0^1 + \gamma_2 \cdot \theta_0^2 + \gamma_0 \cdot \theta_0^0)$

(QR_1) $\gamma_0 \geq 0.5$

(QR_2) $\gamma_0 > 0.5$

Figure 1.13 – Constraint encodings for quantitative reachability

Set of benchmarks		#pIMCs	#nodes	#edges	#intervals			#paramInBounds			#parameters
					min	avg	max	min	avg	max	
HERMAN	N=3	27	8	28	0	7	18	0	3	11	$\{2,5,10\}$
HERMAN	N=5	27	32	244	19	50	87	0	12	38	$\{2,5,10\}$
HERMAN	N=7	27	128	2 188	37	131	236	3	31	74	$\{5,15,30\}$
EGL	L=2; N=2	27	238	253	16	67	134	0	15	57	$\{2,5,10\}$
EGL	L=2; N=4	27	6 910	7 165	696	1 897	3 619	55	444	1 405	$\{2,5,10\}$
EGL	L=4; N=2	27	494	509	47	136	276	3	32	115	$\{2,5,10\}$
EGL	L=4; N=4	27	15 102	15 357	1448	4 068	7 772	156	951	3 048	$\{2,5,10\}$
BRP	M=3; N=16	27	886	1 155	16	64	135	1	15	45	$\{2,5,10\}$
BRP	M=3; N=32	27	1 766	2 307	40	128	275	3	32	129	$\{2,5,10\}$
BRP	M=4; N=16	27	1 095	1 443	22	80	171	0	20	62	$\{2,5,10\}$
BRP	M=4; N=32	27	2 183	2 883	49	164	323	3	39	139	$\{2,5,10\}$
CROWDS	CS=10; TR=3	27	6 563	15 143	1 466	3 036	4 598	57	235	535	$\{5,15,30\}$
CROWDS	CS=5; TR=3	27	1 198	2 038	190	410	652	8	31	76	$\{5,15,30\}$
NAND	K=1; N=10	27	7 392	11 207	497	980	1 416	109	466	1 126	$\{50,100,250\}$
NAND	K=1; N=5	27	930	1371	60	121	183	9	58	159	$\{50,100,250\}$
NAND	K=2; N=10	27	14 322	21 567	992	1 863	2 652	197	866	2 061	$\{50,100,250\}$
NAND	K=2; N=5	27	1 728	2 505	114	217	329	23	101	263	$\{50,100,250\}$

Table 1.14 – Benchmarks composed of 459 pIMCs over 5 families used for verifying qualitative properties

qualitative properties only use linear constraints while the CSPs encodings for verifying the quantitative properties produce quadratic constraints (*i.e.*, non-linear constraints). This produces an order of magnitude between the time complexity for solving the qualitative properties vs the quantitative properties w.r.t. our encodings. Thus, we consider two different benchmarks presented in Table 1.14 and 1.15. In both cases, pIMCs are automatically generated from the PRISM model in a text format inspired from [WTO+11].

For the first benchmark used for verifying qualitative properties, we constructed the pIMCs from existing MCs by randomly replacing some exact probabilities on transitions by (parametric) intervals of probabilities. Our pIMC generator takes 4 arguments: the MC transition function; the number of parameters for the generated pIMC; the ratio of the number of intervals over the number of transitions in the generated pIMC; the ratio of the number of parameters over the number of interval endpoints for the generated pIMC. The benchmarks used are presented in Table 1.14. We selected 5 applications from PRISM [KNP11]: HERMAN - the self-stabilisation protocol of Herman from [KNP12]; EGL - the contract signing protocol of Even, Goldreich & Lempel from [NS06]; BRP - the bounded retransmission protocol from [DJJL01]; CROWDS - the crowds protocol from [Shm04]; and NAND - the nand multiplexing from [NPKS05]. Each one is instantiated by setting global constants (*e.g.*, N for the application HERMAN, L and N for the application EGL) leading to more or less complex MCs. We used our pIMC generator to generate a heterogeneous set of benchmarks: 459 pIMCs with 8 to 15 102 states, and 28 to 21 567 transitions not reduced to $[0,0]$. The pIMCs contain from 2 to 250 parameters over 0 to

Benchmarks		#nodes	#edges	#paramInBounds	#parameters
NAND	K=1; N=2	104	147	82	4
NAND	K=1; N=3	252	364	200	5
NAND	K=1; N=5	930	1 371	726	7
NAND	K=1; N=10	7 392	11 207	5 698	12

Table 1.15 – Benchmarks composed of 4 pIMCs used for verifying quantitative properties

Encoding	Size of the produced CSPs	Boolean var.	Integer var.	Real-number var.	Boolean constr.	Linear constr.	Quadratic constr.
SotA	exponential	no	no	yes	yes	yes	no
$\mathbf{C}_{\exists\mathbf{c}}$	linear	yes	no	yes	yes	yes	no
$\mathbf{C}_{\exists\mathbf{r}}$	linear	yes	yes	yes	yes	yes	no
$\mathbf{C}_{\exists\bar{\mathbf{r}}}$	linear	yes	yes	yes	yes	yes	yes

Table 1.16 – Characteristics of the four CSP encodings **SotA**, $\mathbf{C}_{\exists\mathbf{c}}$, $\mathbf{C}_{\exists\mathbf{r}}$, and $\mathbf{C}_{\exists\bar{\mathbf{r}}}$.

7 772 intervals.

For the second benchmark used for verifying quantitative properties we extended the NAND model from [NPKS05]. The original MC NAND model has already been extended as a pMC in [DJJ$^+$15], where the authors consider a single parameter p for the probability that each of the N *nand* gates fails during the multiplexing. We extend this model to pIMC by considering one parameter for the probability that the initial inputs are stimulated, and we have one parameter per *nand* gate to represent the probability that it fails. We consider 4 pIMCs with 104 to 7 392 states, and 147 to 11 207 transitions not reduced to $[0,0]$. The pIMCs contain from 4 to 12 parameters appearing over 82 to 5 698 transitions.

1.6.2 Constraint modeling

Given a pIMC in a text format our tool produces the desired CSP according to the selected encoding (*i.e.*, one from [DLP16], $\mathbf{C}_{\exists\mathbf{c}}$, $\mathbf{C}_{\exists\mathbf{r}}$, or $\mathbf{C}_{\exists\bar{\mathbf{r}}}$). Recall that our benchmark only consider linear parametric expressions on transitions. The choice of the contraint programming language for implementing a CSP encoding depends on its nature (*e.g.*, the type of the variables: integer vs. continuous, the kind of the contraints: linear vs. non-linear). Table 1.16 summarizes the nature of the encodings where **SotA** stands for the encoding from [DLP16] answering the existential consistency problem. Thus, **SotA**, $\mathbf{C}_{\exists\mathbf{c}}$, and $\mathbf{C}_{\exists\mathbf{r}}$ can be implemented as Mixed Integer Linear Programs (`MILP`) [Vie15] and as Satisfiability Modulo Theory (`SMT`) programs [BSST09] with QF_LRA logic (Quantifier Free Linear Real-number Arithmetic). This logic deals with Boolean combinations of inequations between linear polynomials over real variables. Note that, QF_LRA

Set of benchmarks		avg(#variables)			avg(#constraints)			avg(encod. time)			avg(solv. time)		
		(1)	(2)	(3)	(1)	(2)	(3)	(1)	(2)	(3)	(1)	(2)	(3)
HERMAN	N=3	71	42	42	1258	272	238	0.10	0.07	0.07	0.01	0.01	0.01
HERMAN	N=5	1 031	282	282	51 064	2 000	1 750	1.52	0.08	0.08	0.24	0.03	0.01
HERMAN	N=7	16 402	2 333	2 333	1 293 907	16 483	14 278	50.47	0.13	0.13	5.92	0.28	0.04
EGL	L=2; N=2	462	497	497	4 609	3 917	3 658	0.21	0.08	0.08	0.02	0.04	0.01
EGL	L=2; N=4	13 786	14 081	14 081	138 596	112 349	105 178	5.66	0.44	0.44	0.54	2.15	0.36
EGL	L=4; N=2	958	1 009	1 009	9 560	8 013	7 498	0.36	0.10	0.10	0.04	0.08	0.02
EGL	L=4; N=4	30 138	30 465	30 465	301 866	243 421	228 058	13.03	0.87	0.87	1.26	11.31	0.97
BRP	MAX=3; N=16	68 995	2 047	2 047	738 580	16 063	14 902	32.29	0.12	0.12	3.54	0.21	0.06
BRP	MAX=3; N=32	OM	4 079	4 079	OM	32 047	29 734	OM	0.18	0.18	OM	0.47	0.13
BRP	MAX=4; N=16	103 105	2 544	2 544	1 114 774	19 960	18 511	46.54	0.13	0.13	5.42	0.27	0.08
BRP	MAX=4; N=32	OM	5 072	5 072	OM	39 832	36 943	OM	0.21	0.21	OM	0.63	0.17
CROWDS	CS=10; TR=3	OM	21 723	21 723	OM	165 083	149 923	OM	0.67	0.66	OM	11.48	0.79
CROWDS	CS=5; TR=3	OM	3 253	3 253	OM	25 063	23 008	OM	0.16	0.15	OM	0.39	0.09
NAND	K=1; N=10	87 506	18 732	18 732	888 733	145 108	133 768	152.06	0.56	0.56	3.72	6.21	0.79
NAND	K=1; N=5	6 277	2 434	2 434	62 987	18 098	16 594	10.26	0.12	0.12	0.24	0.25	0.07
NAND	K=2; N=10	169 786	36 022	36 022	1 722 970	279 998	258 298	298.93	1.04	1.04	7.75	31.81	2.06
NAND	K=2; N=5	11 623	4 366	4 366	117 814	33 218	30 580	19.24	0.17	0.17	0.44	0.48	0.13

Table 1.17 – Comparison of sizes, encoding, and solving times for three approaches: (1) **SotA** encoding implemented in SMT, (2) $\mathbf{C}_{\exists \mathbf{c}}$ encoding implemented in SMT, and (3) $\mathbf{C}_{\exists \mathbf{c}}$ encoding implemented in MILP (times are given in seconds).

does not deal with integer variables. Indeed logics mixing integers and reals are harder than those over reals only. However, all integer variables in our encodings can be replaced by real-number variables [6]. Each integer variable x can be declared as a real variable whose real domain bounds are its original integer domain bounds; we also add the constraint $x < 1 \Rightarrow x = 0$. Since we only perform incrementation of x this preserves the same set of solutions (*i.e.*, ensures integer integrity constraints). Finally, due to the non-linear constraints in $\mathbf{C}_{\exists \bar{\mathbf{r}}}$, these encodings are implemented as `SMT` programs [BSST09] with the QF_NRA logic (Quantifier Free Non linear Real-number Arithmetic). We use the same technique as for $\mathbf{C}_{\exists \mathbf{c}}$ and $\mathbf{C}_{\exists \mathbf{r}}$ for replacing integer variables by real-number variables. We chose the programming language Python for implementing our CSP modeler. We do not evaluate any arithmetic expression while generating CSPs, and numbers in the interval endpoints of the pIMCs are read as strings and no trivial simplification is performed while modeling. We do so to avoid any rounding of the interval endpoints when using floating point numbers.

Experiments have been realized on a 2.4 GHz Intel Core i5 processor. Time out has been set to 10 minutes. Memory out has been set to 2Gb. Table 1.17 presents the average size (*i.e.*, the number of variables and the number of constraints) of the instances considered for each set of benchmarks introduced in Table 1.14, as well as the average encoding and solving time for the existential consistency problem using (1) `SMT` **SotA** encoding, (2) `SMT` $\mathbf{C}_{\exists \mathbf{c}}$ encoding, and (3)

6. Note that obtaining integer integrity constraints over real-numbers can be costly.

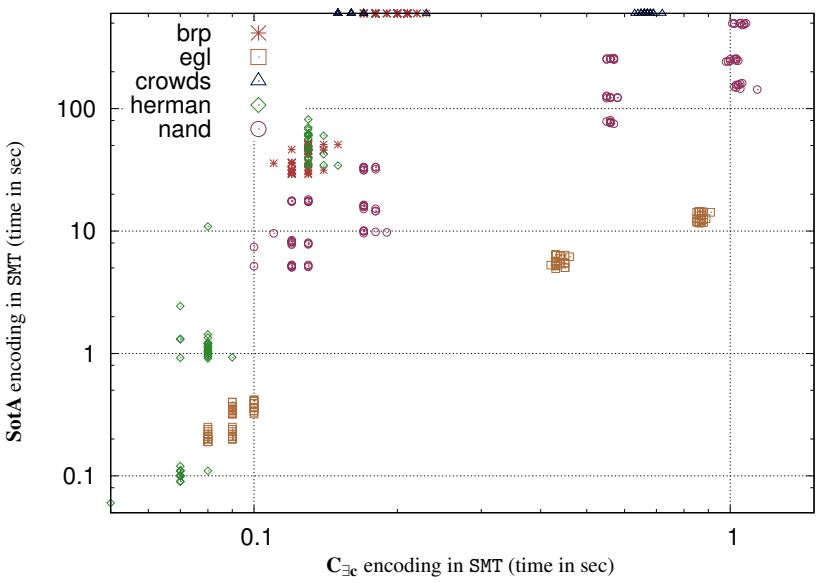


Figure 1.18 – Comparing encoding time for the existential consistency problem

MILP $\mathbf{C}_{\exists\mathbf{c}}$ encoding. First, note that all pIMCs are successfully compiled when using our $\mathbf{C}_{\exists\mathbf{c}}$ encoding while the **SotA** encoding produces out of memory errors for 4 sets of benchmarks: more than 20% of the instances (see OM cells in Table 1.17). We recall that the **SotA** encoding is defined inductively, and that it iterates over the power set of the states. In practice, this implies deep recursions joined with enumeration over the power set of the states. The exponential gain exposed in Section 1.4 is visible in terms of number of variables and constraints in Table 1.17, and in terms of encoding time in Figure 1.18. Each dot in Figure 1.18 corresponds to one instance of our benchmark. While the encoding time ranges between 0 and 1s when using the $\mathbf{C}_{\exists\mathbf{c}}$ encoding, it varies between 0 and 500s when using the **SotA** encoding (if it does not run out of memory).

MILP formulation of logical constraints (*e.g.*, conjunction, disjunction, implication, equivalence) implies the introduction of binary variables called indicator variables [BBF+16]. Each indicator variable is associated to one or more constraints. The valuation of the indicator variable activates or deactivates its associated constraints. We tried to formulate the **SotA** encoding into MILP. Unfortunately, the nested conjunctions and disjunctions imply the introduction of a huge number of indicator variables, leading to giant instances giving bad encoding and solving time. However, since the Boolean variables in $\mathbf{C}_{\exists\mathbf{c}}$ exactly correspond to indicator variables, the MILP formulation of the $\mathbf{C}_{\exists\mathbf{c}}$ encoding does not introduce additional variables or constraints. The difference between $\mathbf{C}_{\exists\mathbf{c}}$ in SMT and $\mathbf{C}_{\exists\mathbf{c}}$ in MILP comes from the encoding of the domains of the continuous variables: in SMT, it requires the use of inequality constraints, *e.g.*, $0 \leq x \leq 1$. The encoding time is the

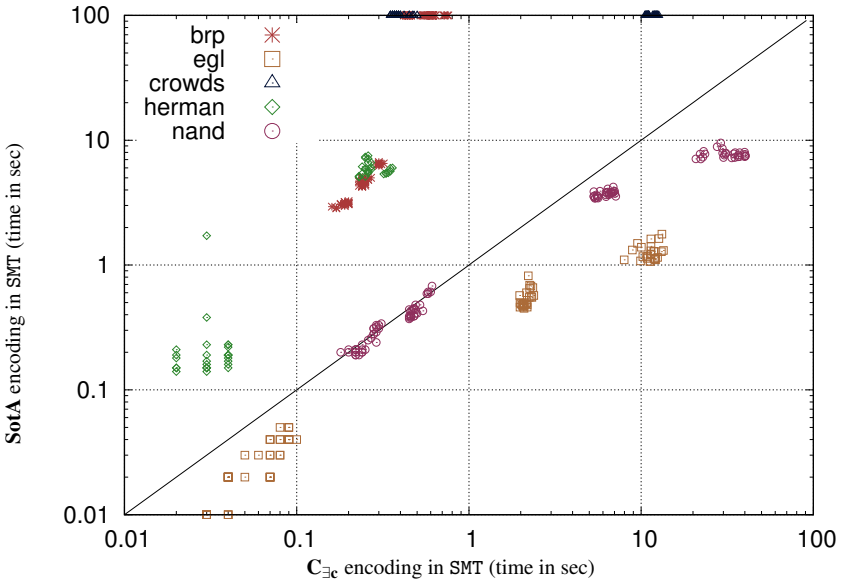


Figure 1.19 – Comparing solving time for the existential consistency problem

same for SMT and MILP $\mathbf{C}_{\exists\mathbf{c}}$ encoding.

1.6.3 Solving

We chose Z3 [DMB08] in its last version (v. 4.4.2) as SMT solver. We chose CPLEX [cpl10] in its last version (v. 12.6.3.0) as MILP solver. Both solvers have not been tuned and we use their default strategies. Experiments have been realized on a 2.4 GHz Intel Core i5 processor. Time out has been set to 10 minutes. Memory out has been set to 2Gb.

Table 1.17 presents the resolution time for the existential consistency problem on our first benchmark using (1) SMT **SotA** encoding, (2) SMT $\mathbf{C}_{\exists\mathbf{c}}$ encoding, and (3) MILP $\mathbf{C}_{\exists\mathbf{c}}$ encoding. While the **SotA** CSPs are larger than the $\mathbf{C}_{\exists\mathbf{c}}$ CSPs, the solving time for the **SotA** CSPs appears to be competitive compared to the solving time for the $\mathbf{C}_{\exists\mathbf{c}}$ CSPs. The scatter plot in Figure 1.19 (logarithmic scale) compares solving times for the SMT **SotA** encoding and SMT $\mathbf{C}_{\exists\mathbf{c}}$ encoding. However when considering the resolution time of the problem (*i.e.*, the encoding time plus the solving time) the $\mathbf{C}_{\exists\mathbf{c}}$ encoding clearly answers faster than the **SotA** encoding. Finally, the comparison between the solving time using SMT $\mathbf{C}_{\exists\mathbf{c}}$ encoding and MILP $\mathbf{C}_{\exists\mathbf{c}}$ encoding is illustrated in Figure 1.20. It shows that Both the loss of safety (passing from real numbers with Z3 SMT resolution to floating point numbers with CPLEX MILP resolution) and the fact that CPLEX is highly optimized for MILP problems whereas Z3 is a "generic" solver lead to a non negligible gain in terms of resolution time (near to an exponential gain in our benchmark). Indeed

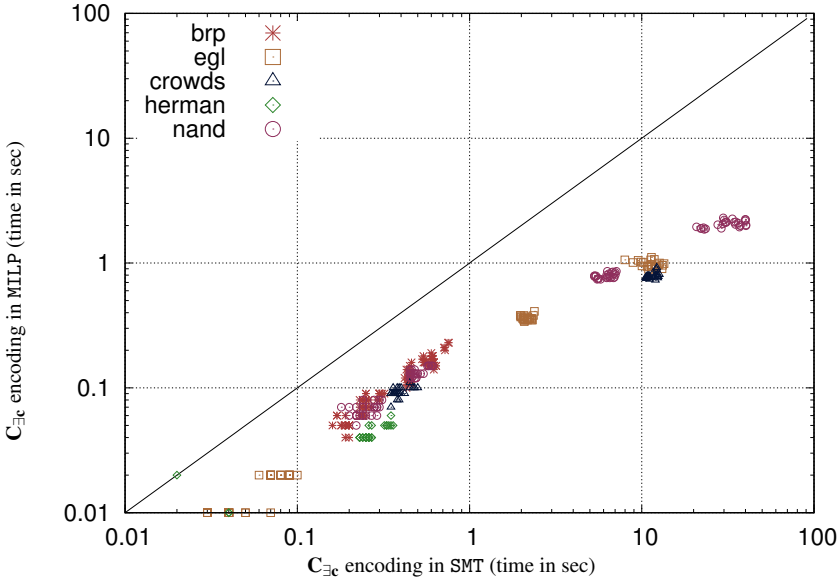


Figure 1.20 – Comparing solving time between SMT and MILP formulations

Benchmark	pIMC			$\mathbf{C}_{\exists\mathbf{c}}$			$\mathbf{C}_{\exists\mathbf{r}}$			$\mathbf{C}_{\exists\bar{\mathbf{r}}}$		
	#states	#trans.	#par.	#var.	#cstr.	time	#var.	#cstr.	time	#var.	#cstr.	time
NAND K=1; N=2	104	147	4	255	1 526	0.17s	170	1 497	0.19s	296	2 457	69.57s
NAND K=1; N=3	252	364	5	621	3 727	0.24s	406	3 557	0.30s	703	5 828	31.69s
NAND K=1; N=5	930	1 371	7	2 308	13 859	0.57s	1 378	12 305	0.51s	2 404	20 165	T.O.
NAND K=1; N=10	7 392	11 207	12	18 611	111 366	9.46s	9 978	89 705	13.44s	17 454	147 015	T.O.

Table 1.21 – Comparison of solving times between qualitative and quantitative encodings.

the SMT $\mathbf{C}_{\exists\mathbf{c}}$ encoding requires 50 seconds to complete the solving process while the MILP $\mathbf{C}_{\exists\mathbf{c}}$ encoding needs less than 5 seconds for the same instances.

Table 1.21 summarizes the results w.r.t. our second benchmark: the pIMC sizes (in terms of states, transitions, and parameters), the CSP sizes (in terms of number of variables and constraints), and the resolution time using the Z3 solver. Note first that we perform pre-processing when verifying reachability properties: using a simple graph analysis, we eliminate some states that cannot, trivially, reach the goal states. This explains why $\mathbf{C}_{\exists\mathbf{r}}$ has less variables and constraints than $\mathbf{C}_{\exists\mathbf{c}}$. Finally, note the order of magnitude between the resolution time required for solving the qualitative properties vs. the quantitative properties w.r.t. our encodings. Indeed, we did not succeed in solving pIMCs with more than 300 states and 400 transitions for quantitative properties while we verified pIMCs with more than 10 000 states and 20 000 transitions in the qualitative context.

1.7 Conclusion

In this chapter, we have compared several Markov Chain abstractions in terms of succinctness and we have shown that Parametric Interval Markov Chain is a strictly more succinct abstraction formalism than other existing formalisms such as Parametric Markov Chains and Interval Markov Chains. In addition, we have proposed constraint encodings for checking several properties over pIMC. In the context of qualitative properties such as existential consistency or consistent reachability, the size of our encodings is significantly smaller than other existing solutions. In the quantitative setting, we have compared the three semantics for IMCs and pIMCs and showed that the semantics are equivalent with respect to quantitative reachability properties. As a side effect, this result ensures that all existing tools and algorithms solving reachability problems in IMCs under the once-and-for-all semantics can safely be extended to the IMDP and at-every-step semantics with no changes. Based on this result, we have then proposed CSP encodings addressing quantitative reachability in the context of pIMCs regardless of the chosen semantics. Finally, we have developed a prototype tool that automatically generates our CSP encodings and that can be plugged to any constraint solver accepting the SMT-LIB format as input.

1.8 Perspectives

The results presented in this chapter could be extended in several manners. First, our tool for pIMC verification could be extended in order to manage other, more complex, properties (*e.g.*, supporting the LTL language in the spirit of what Tulip [WTO+11] does). However, as shown in Section 1.5.1, the techniques we use, based on the equivalence of the three IMC semantics for reachability properties, cannot be easily extended to more general properties. Some work is therefore needed to find alternative techniques when considering other properties than reachability.

In this line of work, we have investigated the reward-bounded reachability problem in the context of Parametric Markov Reward Models (*i.e.*, pIMC where each state is associated to a reward). We have shown that in the context of model checking rPCTL, an extension of PCTL where each path formula is equipped with a specification of a bound on the accumulated reward, the three semantics (once-and-for-all, IMDP and at-every-step) are not equivalent. However, we have also shown that the at-every-step and IMDP semantics are equivalent when restricted to reward-bounded reachability properties. Although this does not provide the same simplification

as if they were equivalent to the once-and-for-all semantics (allowing to preserve the structure), we have still proposed an algorithm for the verification of reward-bounded reachability properties using a reduction to the model checking problem for parametric Markov Chains. These results have been presented in a paper that is under evaluation at the moment.

The consistency and reachability problems could also be extended to other type of interval-based specifications. In particular, we have also studied these problems in the context of parametric Interval Probabilistic Timed Automata. In this context, parameters can range both on timing constraints (as is the case in parametric Timed Automata) or on interval endpoints (as is the case here). In [AD16, ADF20], we have restricted ourselves to parameters only appearing in timing constraints, leaving the intervals free of parameters. With this restriction, we used the results presented in this chapter to propose an algorithm solving the problem in the non-parametric case. We have shown that the problem is undecidable in the parametric setting, but we have proposed a syntactic condition on the use of parameters that ensures decidability along with a construction that solves the problem in this case. Considering the full class where parameters can range both on timing parameters and interval endpoints remains to be done.

Another line of extension could be to consider a parametric extension of a more expressive formalism than Interval Markov Chains. For instance, it would be natural to consider a parametric extension of Constraint Markov Chains [CDL$^+$11]. Our intuition is that the results presented here would naturally extend to this setting. The only difficulty would be to make sure that the equivalence result on the three semantics is still valid in this context.

Finally, we have only focused until now on existential problems, *i.e.*, determining whether there exists parameter values ensuring certain properties. We have also proposed algorithms and techniques for synthesizing solutions to these problems. However, a technique for representing and analyzing the set of *all solutions* to these problems still eludes us, and would be of particular importance in the context of understanding or optimizing the systems under study.

CHAPTER 2

A PROBABILISTIC EXTENSION FOR EVENT-B

The work we present in this chapter was achieved during **Mohamed Amine Aouadhi**'s thesis. This work was initiated from the observation that many software systems are developed using specification and refinement techniques using proof-based modeling formalisms such as B [Abr05], Event-B [Abr10], Z [Spi88] and their associated toolsets. Although uncertainty can be taken into account in such models, the only way of expressing it is through non-determinism. As explained in the previous chapter, uncertainty is a very important notion when modeling complex systems and needs to be taken into account inside the models in order to make verification and analysis of those systems more accurate. As also explained before, uncertainty can be expressed in other, more informative ways, than non-determinism. In particular, the use of probabilities inside the modeling formalism is a way of being more precise when modeling certain kinds of uncertainties than limiting ourselves to non-determinism.
To the best of our knowledge, few attempts had been made at extending the Event-B formalism with probabilities when we started this line of work. In particular, the existing attempts had been limited in the manner in which probabilistic information could be included in Event-B models. In this work, we instead propose a probabilistic extension of Event-B where probabilities can appear in any place where choices are available, and can be introduced in the models in a progressive manner through probabilistic refinement. This chapter is mostly based on [ADL17, ADL19b]. For the sake of conciseness, some of the results obtained in [ADL19b] as well as detailed proofs have been left out of this document. However, the interested reader can refer to [ADL19b] for details.

2.1 Introduction

As systems become more and more complex, with randomised algorithms [MR10], probabilistic protocols [ACM03] or failing components, it is necessary to add new modeling features in order to take into account complex system properties such as reliability [Vil92], responsiveness [CS88, TRF03], continuous evolution, energy consumption etc. One of these features is

probabilistic reasoning to introduce uncertainty in a model or to mimic randomised behaviour. Probabilistic modeling formalisms have therefore been developed in the past, mainly extending automata-based formalisms [Sto02, Put14]. One such probabilistic modeling formalism has been presented in Chapter 1. Abstraction [Kat07, DGVJ12], refinement [JL91] and model checking algorithms [BDA95, BK08] have been successfully studied in this context. However, the introduction of probabilistic reasoning in proof-based modeling formalisms has been, to the best of our knowledge, quite limited [HA12, ST96, Hoa05, Gol98, BFG$^+$14, HMM05, APM09, Hur03]. Translations from proof-based models are always possible. However, the use of automata-based verification in this context is inconvenient due to the state-space explosion in the translation.

Event-B [Abr10] is a proof-based formal method used for discrete systems modeling. It is equipped with *Rodin* [ABH$^+$10], an open toolset for modeling and proving systems. This toolset can easily be extended, which makes of Event-B a good candidate for introducing probabilistic reasoning in a proof-based modeling formalism. The development process in Event-B is based on stepwise refinement: systems are typically developed progressively using an ordered sequence of models, where each model contains more details than its predecessor. Refinement allows a step-by-step description of the behaviour of systems, providing an efficient way to give a detailed description of their behaviour.

So far, several research works have focused on the extension of Event-B to allow the expression of probabilistic information in Event-B models. In [MHA05], Abrial *et al.* have summarised the difficulties of embedding probabilities into Event-B. This paper suggests that probabilities need to be introduced as a refinement of *non-determinism*. In Event-B, non-determinism occurs in several places such as the choice between enabled events in a given state, the choice of the parameter values in a given event, and the choice of the value given to a variable through some non-deterministic assignments. To the best of our knowledge, the existing works on extending Event-B with probabilities have mostly focused on refining non-deterministic assignments into probabilistic ones while other sources of non-determinism have been left untouched. In [HH07], Hallerstede *et al.* focus on a qualitative aspect of probability. They refine non-deterministic assignments into *qualitative* probabilistic assignments where the actual probability values are not specified, and adapt the Event-B semantics and proof obligations to this new setting. In [Yil10], the same authors study the refinement of qualitative probabilistic Event-B models and propose a tool support inside Rodin. Other works [TTL09, TTL15, TTL10] have extended this approach by refining non-deterministic assignments into *quantitative* probabilistic assignments where, unlike in [HH07], the actual probability values are specified. This new proposition is then exploited in order to assess several system properties such as reliability and responsiveness.

Unfortunately, other sources of non-determinism than assignments have been left untouched, although the authors argue that probabilistic choice between events or parameter values can be achieved by transformations of the models that embed these choices inside probabilistic assignments. While this is unarguably true, such transformations are not trivial and greatly impede the understanding of Event-B models. Moreover, these transformations would need to be included in the refinement chain when designers need it, which would certainly be counter-intuitive to engineers.

In this chapter, we extend these works by proposing a probabilistic extension of Event-B and presenting some ways of introducing probabilistic reasoning within Event-B. As the design process within Event-B is based on refinement, we propose to provide a standard description of a system by a set of models related by refinement. According to this modeling process, probabilities can be introduced in several manners that are illustrated in Figure 2.1.

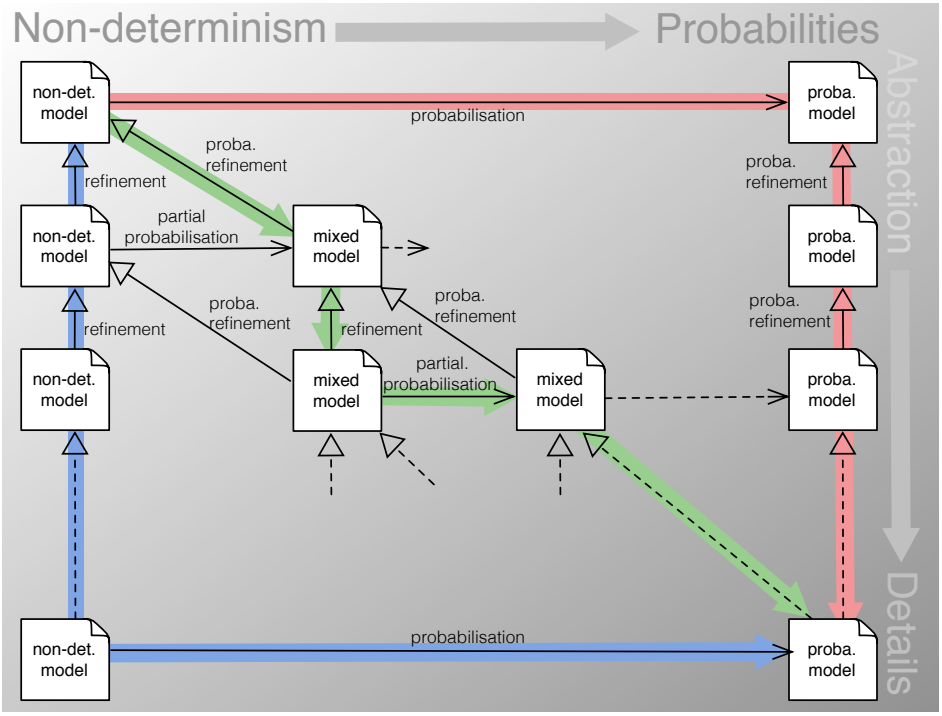


Figure 2.1 – How to introduce probabilities within Event-B

In this figure, the vertical axis represents the introduction of more details in the model while the horizontal axis represents the introduction of probabilistic information. Several types of models stand out:

— Models on the left side of the picture are *standard* Event-B models that only contain

non-deterministic choices but no probabilistic information;

— Models on the right side of the picture are *fully probabilistic* Event-B models where all choices are probabilistic;

— Models in the centre of the picture are *mixed* Event-B models, where both non-deterministic and probabilistic choices are present.

Depending on the system that is being developed, the development process could always stay on the left side (when the system does not have any probabilistic aspect), and therefore end in the bottom left of the picture; or the development process could move to the right side and either end in the bottom right when the system is fully probabilistic or in the middle when the system contains both probabilistic and non-deterministic aspects.

In any case, there are many ways both to add standard (non-deterministic) details in the model and to add probabilistic information. Fig. 2.1 provides three generic development processes (green, red and blue), which we detail below.

Assuming the model under development is fully probabilistic, one could consider starting with an abstract non-deterministic version of the model, then progressively refining it in a standard way until a satisfying level of details is achieved. Once enough details have been introduced, all non-deterministic choices can be refined into probabilistic choices in one shot (this last step is called *probabilisation*). This whole process is depicted in blue in Fig. 2.1.

Obviously, one could also consider starting with the probabilisation step, therefore obtaining an abstract fully probabilistic model. From this model, details can then be introduced through *probabilistic refinement*. This whole process is depicted in red in Fig. 2.1. While the fully probabilistic counterpart of the standard Event-B refinement still eludes us at this point, we nevertheless propose some restricted refinement steps for fully probabilistic systems through context refinement and the introduction of new probabilistic events.

Finally, the designer could decide to interleave the introduction of new details in the model with the introduction of probabilistic information. In this context, intermediate models are mixed models and one has to consider the standard refinement of mixed models, the *partial probabilisation* operation (that only turns some of the non-deterministic choices into probabilistic choices), the introduction of probabilistic events in a standard (non-deterministic) model and the introduction of standard (non-deterministic) events in a probabilistic or mixed model. That last development process is depicted in green in Fig. 2.1.

This chapter summarizes the results obtained during Mohamed Amine Aouadhi's thesis and is based on [ADL17, ADL19b]. In these papers, we have provided the scientific foundations in order to allow all the design possibilities presented above in the Event-B framework. In particular,

we have proposed some new syntactic elements for writing probabilistic and mixed Event-B models in the Event-B framework. The consistency of such models is then expressed, as in standard Event-B, in terms of proof obligations. In order to prove the correctness of our approach, we have shown that the operational semantics of such models can be expressed in terms of (potentially infinite-state) Markov Chains – for fully probabilistic models – and (potentially infinite-state) Markov Decision Processes – for mixed Event-B models – therefore resembling the LTS operational semantics of standard Event-B models.

In this chapter, we propose several operations for introducing details and probabilistic aspects in probabilistic Event-B models. [1] In particular, we focus on the introduction of new probabilistic events in a given model. In the standard Event-B setting, *convergence* is a required property for proving a refinement steps as soon as new events are introduced in the model. The counterpart property in the probabilistic setting is *almost-certain convergence*, which has already been studied in [MM06, Hoa05] in the context of probabilistic programs and the standard B method, and in [Hoa14, HH07] in the context of non-deterministic Event-B models with only probabilistic assignments. While the authors of [MM06, Hoa05] propose hypotheses under which probabilistic while loops almost-certainly converge, these hypotheses cannot be directly applied to our setting as they would require a translation from the probabilistic Event-B setting to the standard probabilistic B setting which is not trivial. In addition, some new conditions would need to be exhibited in the probabilistic Event-B setting that ensure that the hypotheses on the standard probabilistic B setting are met. Instead, have chosen to exhibit conditions at the probabilistic Event-B level and we show that these conditions ensure almost-certain convergence of the operational semantics of the model. On the other hand, [Hoa14, HH07] focus on almost-certain convergence at the probabilistic Event-B level for probabilistic Event-B models where probabilities only appear inside probabilistic assignments, but cannot appear in the choice between enabled events or in the choice of parameter values. However, we show that the proof obligations developed in this context are not sufficient for our models. We therefore propose new sufficient conditions, expressed in terms of proof obligations, for the almost-certain convergence of a set of fully probabilistic events. While the conditions we exhibit are more constrained than those from [HH07] concerning events and parameters, they are also less restrictive concerning probabilistic assignments.

Finally, some of the results presented in this chapter have been implemented in a prototype plugin for Rodin, which we briefly present at the end of the chapter.

1. Some aspects such as the probabilisation of standard Event-B models or the study of mixed Event-B models are deliberately left out in order to keep this document to a manageable size. Nevertheless, these topics have also been studied and are detailed in [ADL19b].

2.2 Background

We start by introducing notations for (probabilistic) transition systems and basic elements of the Event-B method that will be used throughout the chapter.

2.2.1 Transition systems

In the following, recall that $\mathsf{Dist}(S)$ denotes the set of distributions over a given set S, *i.e.*, the set of functions $\delta : S \to [0,1]$ such that $\sum_{s \in S} \delta(s) = 1$. Remark that the definitions we give here are slightly different from the definitions given in Chapter 1. Indeed, in the context of Event-B, the set of atomic propositions that label states in our probabilistic models may depend on the corresponding Event-B model and therefore have to be included in the following definitions. Moreover, probabilistic transition systems in this context (such as Markov Chains), need to take into account action names that label transitions.

Labelled Transition System [BK08] A labelled transition system (LTS for short) is a tuple $\mathcal{M}$=(S, $Acts$, s_0, $\rightarrow$, AP, L) where S is a set of states, $s_0 \subseteq S$ is the initial state, $Acts$ is a set of actions, $\rightarrow \subseteq S \times Acts \times S$ is a transition relation AP is a set of atomic propositions, and $L : S \rightarrow 2^{AP}$ is a labelling function.

Probabilistic Labelled Transition System [BK08] A Probabilistic Labelled Transition System (PLTS for short) is a tuple $\mathcal{M}$=(S, s_0, $Acts$, P, AP, L) where S is a set of states, $s_0 \in S$ is the initial state, $Acts$ is a set of actions, AP is a set of atomic propositions, L: $S \rightarrow AP$ is a labelling function, and P: $S \times Acts \times S \rightarrow [0,1]$ is the transition probability function. If for each state $s \in S$ we have $\sum_{s' \in S, a \in Acts} P(s,a,s') = 1$, then the PLTS is a *Discrete Time Markov Chain* (as introduced in Chapter 1).

2.2.2 Event-B

Event-B [Abr10] is a formal method used for the development of complex discrete systems. Systems are described in Event-B by means of models. For the sake of simplicity, we assume in the rest of the chapter that an Event-B model is expressed by a tuple M=($\bar{v}$,I($\bar{v}$),V($\bar{v}$),Evts, Init) where $\bar{v}$= $\{\mathsf{v}_1 \ldots \mathsf{v}_n\}$ is a set of variables, I($\bar{v}$) is an invariant, V($\bar{v}$) is an (optional) variant used for proving the convergence of the model, Evts is a set of events and Init $\in$ Evts is the initialisation

event. The invariant $I(\bar{v})$ is a conjunction of predicates over the variables of the system specifying properties that must always hold.

Events An event has the following structure (see Figure on the side), where e_i is the name of the event, $\bar{t} = \{t_1 \ldots t_n\}$ represents the (optional) set of parameters of the event, $G_i(\bar{t},\bar{v})$ is the (optional) guard of the event and $S_i(\bar{t},\bar{v})$ is the action of the event. An event is *enabled* in a given valuation of the variables (also called a configuration) if and only if there exists a parameter valuation such that its guard $G_i(\bar{t},\bar{v})$ is satisfied in this context. The action $S_i(\bar{t},\bar{v})$ of an event may contain several assignments that are executed in parallel. An assignment can be expressed in one of the following forms:

$e_i \mathrel{\widehat{=}}$
any $\bar{t}$ **where**
$G_i(\bar{t},\bar{v})$
then
$S_i(\bar{t},\bar{v})$
end

— **Deterministic assignment**: $x := E(\bar{t},\bar{v})$ means that the expression $E(\bar{t},\bar{v})$ is assigned to the variable x.

— **Predicate (non-deterministic) assignment**: $x :| Q(\bar{t},\bar{v},x,x')$ means that the variable x is assigned a new value x' such that the predicate $Q(\bar{t},\bar{v},x,x')$ is satisfied.

— **Enumerated (non-deterministic) assignment**: $x :\in \{E_1(\bar{t},\bar{v}) \ldots E_n(\bar{t},\bar{v})\}$ means that the variable x is assigned a new value taken from the set $\{E_1(\bar{t},\bar{v}) \ldots E_n(\bar{t},\bar{v})\}$, where E_i are expressions.

Before-after predicate and semantics The formal semantics of an assignment is described by means of a before-after predicate (BA) $Q(\bar{t},\bar{v},x,x')$. This BA describes the relationship between the values of the variable before (x) and after (x') the execution of an assignment. Before-after predicates are as follows:

— the BA of a deterministic assignment is $x' = E(\bar{t},\bar{v})$,

— the BA of a predicate assignment is $Q(\bar{t},\bar{v},x,x')$, and

— the BA of an enumerated assignment is $x' \in \{E_1(\bar{t},\bar{v}) \ldots E_n(\bar{t},\bar{v})\}$.

Recall that the action $S_j(\bar{t},\bar{v})$ of a given event e_j may contain several assignments that are executed in parallel. Assume that $v_1 \ldots v_i$ are the variables assigned in $S_j(\bar{t},\bar{v})$ – variables $v_{i+1} \ldots v_n$ are thus not modified – and let $Q(\bar{t},\bar{v},v_1,v'_1) \ldots Q(\bar{t},\bar{v},v_i,v'_i)$ be their corresponding BA. Then the BA $S_j(\bar{t},\bar{v},\bar{v}')$ of the event action $S_j(\bar{t},\bar{v})$ is:

$$S_j(\bar{t},\bar{v},\bar{v}') \mathrel{\widehat{=}} Q(\bar{t},\bar{v},v_1,v'_1) \wedge \ldots \wedge Q(\bar{t},\bar{v},v_i,v'_i) \wedge (v'_{i+1}=v_{i+1}) \wedge \ldots (v'_n=v_n)$$

Proof obligations The consistency of a standard Event-B model is characterized by means of *proof obligations* (POs) which must be discharged. Discharging all the necessary POs allows to prove that the model is sound with respect to some behavioural semantics. Formal definitions of standard Event-B POs are given in [Abr10]. In the following, we only recall the most important of them: (event/INV) for *invariant preservation*, which states that the invariant still holds after the execution of each event in the Event-B model M. Given an event e_i with guard $\mathsf{G}_i(\bar{t},\bar{v})$ and action $\mathsf{S}_i(\bar{t},\bar{v})$, this PO is expressed as follows:

$$\mathsf{I}(\bar{v}) \ \wedge \mathsf{G}_i(\bar{t},\bar{v}) \ \wedge \mathsf{S}_i(\bar{t},\bar{v},\bar{v}') \vdash \mathsf{I}(\bar{v}') \qquad \text{(event/INV)}$$

Operational semantics As established in [BC00], the semantics of an Event-B model $\mathsf{M}=(\bar{v},\mathsf{I}(\bar{v}),\mathsf{V}(\bar{v}),\mathsf{Evts},\mathsf{Init})$ is expressed in terms of a labelled transition system (LTS) $\mathcal{M}=(S, s_0, Acts, T, AP, L)$ where S is a set of states, each state in S being uniquely identified by its label; $s_0 \in S$ is the initial state obtained by executing the *Init* event; $Acts$ is the set of actions (event names); AP is the set of atomic propositions: a set of predicates that correspond to the valuations of $\bar{v}$ and satisfy the invariant $\mathsf{I}(\bar{v})$; L: $S \rightarrow AP$ is a labelling function that provides the valuations of the variables $\bar{v}$ in a given state; and $T \subseteq S \times Acts \times S$ is the transition relation corresponding to the actions of the events of M.

2.2.3 Refinement in Event-B

In Event-B, the process of modeling systems is based on the theory of refinement. Many research work have focused in the development of the theory of refinement [BvW89, Bac89, MM06]: it appears from the literature that refinement is used in two related concepts in computer science. The first one considers refinement as a top-down program development methodology when the system is firstly described by an abstract specification and progressively refined by other ones. Each abstract specification will be more detailed and new details can be introduced during refinement. The second concept concerns the preservation of correctness between abstract and refined specifications.

Event-B refinement supports both concepts: it is a mechanism for introducing details about the static and dynamic properties of a model while preserving correctness. For the static part, refinement in Event-B allows a detailed description of the state space by the introduction of new variables (*i.e.*, data/context refinement [HHS86, DREB98]). Concerning the dynamic aspects, refinement in Event-B also allows a more detailed description of the execution of the system by adding new events processing the new introduced variables or by giving more details on the

events of an abstract model. For more details on the theory of refinement in Event-B, we refer the reader to the Action Systems formalism [Bac89] which has inspired the development of Event-B. In Event-B, under a number of conditions expressed as proof obligations, a concrete model N=($\bar{w}$, J($\bar{v}$,$\bar{w}$),Evts$_c$, Init $_c$) may refine an abstract model M=($\bar{v}$,I($\bar{v}$), V($\bar{v}$),Evts$_a$, Init). In this case, $\bar{w}$ is a set of variables containing some (preserved) variables of the abstract model and some new variables introduced by refinement, J($\bar{v}$,$\bar{w}$) is an invariant that must provide a relation between the (removed) abstract variables ($\bar{v}$) and the new concrete variables ($\bar{w}$), Evts$_c$ is the set of concrete events that contains both the refined events (Evts$_c$ $\cap$ Evts$_a$) and the new events introduced by refinement (Evts$_c$ $\setminus$ Evts$_a$), and Init $_c$ is the concrete initialisation event.

Events Each abstract event from the set Evts$_a$ can be refined by one or more concrete events. Moreover, several events from the abstract set Evts$_a$ can be refined a single one. The first case is called *splitting* while the second one is called *merging*.
In this chapter, we do not address the cases of event *splitting* and *merging*, we only consider the refinement of an abstract event e$_a$ by only one concrete event e$_c$ as follows. We remark that the event e$_c$ may contain one more component Wi$_c$($\bar{t}$,$\bar{u}$,$\bar{v}$,$\bar{v}$',$\bar{w}$,$\bar{w}$') which denotes a witness. A witness links the abstract parameters $\bar{t}$ and the abstract variables $\bar{v}$' to concrete parameters $\bar{u}$ and concrete variables $\bar{w}$'.

e$_a$ $\widehat{=}$
any $\bar{t}$ **where**
G$_a$($\bar{t}$,$\bar{v}$)
then
S$_a$($\bar{t}$,$\bar{v}$)
end

e$_c$ $\widehat{=}$
refines e$_a$
any $\bar{u}$ **where**
G$_c$($\bar{u}$,$\bar{w}$)
with
Wi$_c$($\bar{t}$,$\bar{u}$,$\bar{v}$,$\bar{v}$',$\bar{w}$,$\bar{w}$')
then
S$_c$($\bar{u}$,$\bar{w}$)
end

As the Event-B refinement process allows a more detailed description of the execution of the system, it is necessary to introduce new events (Evts$_c$ $\setminus$ Evts$_a$) which characterize the evolution of the new added variables during refinement.

Proof obligations In Event-B refinement, the behaviour of the concrete model must be compatible with the behaviour of the abstract one. This constraint is verified and maintained by some proofs obligations dedicated to refinement. All the refinement POs are presented in [Abr10]. In the following, we recall only the most important ones.
For the refinement of an abstract event e$_a$ by a concrete event e$_c$, the two following POs must be satisfied:

(1) **Guard strengthening.** The guard of e$_c$ is as least as strong as the guard of e$_a$:

$$I(\bar{v}) \wedge J(\bar{v},\bar{w}) \wedge G_c(\bar{u},\bar{w}) \wedge Wi_c(\bar{t},\bar{u},\bar{v},\bar{v}',\bar{w},\bar{w}') \vdash G_a(\bar{t},\bar{v}) \quad \text{(grd/STRENGTH)}$$

(2) **Simulation.** The action of e$_c$ simulates the action of e$_a$:

$$I(\bar{v}) \wedge J(\bar{v},\bar{w}) \wedge G_c(\bar{u},\bar{w}) \wedge S_c(\bar{u},\bar{w},\bar{w}') \vdash S_a(\bar{t},\bar{v},\bar{v}') \quad \text{(act/SIM)}$$

The last ensures that when a concrete event is executed, what it does is not contradictory with what the abstract event does.

When introducing new events during refinement, it is then necessary to show that their introduction cannot prevent the system from behaving as specified in the abstract model. In particular, it is necessary to show that such new events are **convergent**, in the sense that they cannot keep control indefinitely: at some point the system has to stop executing new events in order to follow the behaviour specified in its abstract model.
In order to prove that a set of events is convergent in Event-B, we have to introduce a natural number expression $\mathsf{V}(\bar{v})$, called a *variant*, and show that all convergent events strictly decrease the value of this variant. As a consequence, when the variant hits zero, it is guaranteed that no convergent event can be performed. That leads to two POs to be discharged:

(1) **Numeric variant.** Under the guard $\mathsf{G}_i(\bar{t},\bar{v})$ of each convergent event e_i, the variant $\mathsf{V}(\bar{v})$ is greater or equal to 0.

$$\mathsf{I}(\bar{v}) \ \wedge \ \mathsf{G}_i(\bar{t},\bar{v}) \ \vdash \ \mathsf{V}(\bar{v}) \in \mathsf{NAT} \qquad \text{(event/var/NAT)}$$

(2) **Convergence.** The action $\mathsf{S}_i(\bar{t},\bar{v})$ of each convergent event e_i *must always* decrease the variant $\mathsf{V}(\bar{v})$.

$$\mathsf{I}(\bar{v}) \ \wedge \ \mathsf{G}_i(\bar{t},\bar{v}) \ \vdash \ \forall\, \bar{v}'.\ \ \mathsf{S}_i(\bar{t},\bar{v},\bar{v}') \Rightarrow \mathsf{V}(\bar{v}') < \mathsf{V}(\bar{v}) \qquad \text{(event/VAR)}$$

2.3 Running example: Simple peer-to-peer protocol

We now introduce a running example, based on a simplified scenario of a peer-to-peer protocol inspired from Bittorent [bit]. The description of the complete case study can be found in [pri]. The model considers a set of N clients trying to download a file that has been partitioned into K blocks. Initially, no block has been downloaded by any client. The protocol ends when all the clients have successfully downloaded all the blocks.

Initial protocol model The model $\mathsf{P2P}_1$ given in Fig 2.2 presents an abstract Event-B specification of the protocol. It represents a general abstraction of the behaviour of the protocol with no details included. At this level, the state of the protocol is described by means of one variable DB. This variable contains a matrix which indicates for each client c∈1..N and each block b∈1..K whether the client has already downloaded the block (DB(c↦b)=finished) or not (DB(c↦bc=empty). Initially, no block has been downloaded by any client.

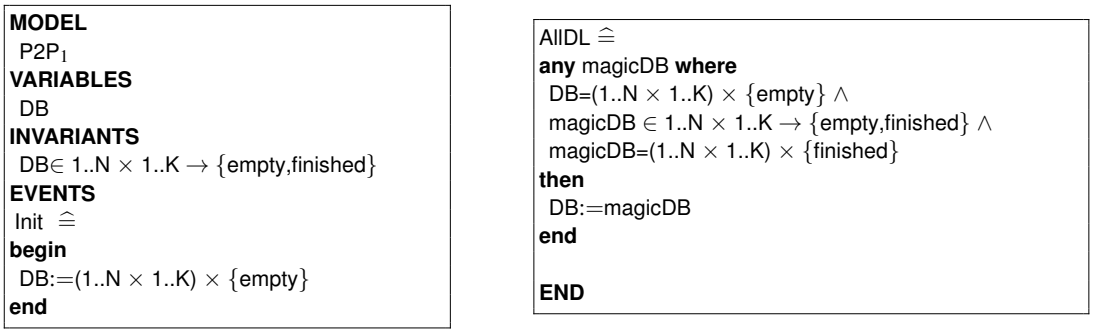


Figure 2.2 – Initial Event-B model of the simple P2P protocol

At this level of abstraction, we only consider one event (AllDL) describing in one statement the whole execution of the protocol. magicDB is a parameter chosen in such a way that for all client c $\in$ 1..N and block b $\in$ 1..K, we have magicDB(c$\mapsto$b)=finished. The substitution DB := magicDB corresponds to the (somehow magical) download of all the blocks by all the clients in one shot. Notice that in reality, the download of all the blocks of the file by all the clients is not done in one shot. It is instead made gradually by successive attempts. Introducing these attempts is the purpose of the first refinement, which we present hereafter.

Step-by-step download We now present a first refinement of the protocol. For this purpose, we enlarge the set of variables and events. The resulting model is presented in Fig 2.3. We introduce a new variable DBin that contains a matrix which represents the state of download of each block at each iteration of the model. For each client and each block, the corresponding state could be finished – indicating that the client has successfully downloaded the block; incoming – indicating that the client is currently trying to download the block; or empty – indicating that the download of the block has not yet started. Initially, as in the abstract model, no block has been downloaded by any client, therefore DBin:= (1..N $\times$ 1..K) $\times$ {empty}. Furthermore, we impose that each client c is not currently trying to download more than one block c, as indicated in the invariant in Fig. 2.3.

To model the download process in a step by step manner, we introduce two new events: Start1DL and Finish1DL. In the event Start1DL, a client c and a block b are chosen in a non-deterministic manner in such a way that the download of the block b by client c has not yet started; furthermore, the considered client c is not currently trying to download another block k; then, the client starts to download the block (DBin(c$\mapsto$b):=incoming). The event Finish1DL models that a client c terminates the downloading of a block b in a similar manner. The events Start1DL and Finish1DL

MODEL
P2P$_2$
REFINES
P2P$_1$

VARIABLES
DB
DBin

INVARIANTS
DBin $\in$ 1..N $\times$ 1..K $\rightarrow$ {empty,incoming,finished} $\wedge$
$\forall$ c . (c $\in$ 1..N $\Rightarrow$
card({b | b $\in$ 1..K $\wedge$ DBin(c$\mapsto$b)=incoming}) $\leq$ 1)

VARIANT
2 $\times$ N*K
$-$ 2 $\times$ card({c$\mapsto$b |n$\in$1..N $\wedge$ b $\in$ 1..K $\wedge$ DBin(c$\mapsto$b) = finished})
$-$ card({c$\mapsto$b |n$\in$1..N $\wedge$ b $\in$ 1..K $\wedge$ DBin(c$\mapsto$b) = incoming})

EVENTS

Init $\widehat{=}$
begin
DB:=(1..N $\times$ 1..K) $\times$ {empty} ||
DBin:=(1..N $\times$ 1..K) $\times$ {empty}
end

DLFinished $\widehat{=}$
refines AllDL
when
DB=(1..N $\times$ 1..K) $\times$ {empty} $\wedge$
DBin=(1..N $\times$ 1..K) $\times$ {finished}
then
DB:=DBin
end

Start1DL $\widehat{=}$
any c, b **where**
c $\in$ 1..N $\wedge$ b $\in$ 1..K $\wedge$
DBin(c$\mapsto$b)=empty $\wedge$
card({k | k $\in$ 1..K $\wedge$ DBin(c$\mapsto$k)=incoming})=0
then
DBin(c$\mapsto$b):=incoming
end

Finish1DL $\widehat{=}$
any c, b **where**
c $\in$ 1..N $\wedge$ b $\in$ 1..K $\wedge$
DBin(c$\mapsto$b)=incoming
then
DBin(c$\mapsto$b):=finished
end
END

Figure 2.3 – First refinement of the simple P2P protocol: step-by-step download

are activated until we cannot find any pair (c,b) such that DBin(c$\mapsto$b)=empty.

The event DLFinished refines the event AllDL. It is now enabled when DBin= (1..N $\times$ 1..K) $\times$ {finished}, i.e, when all the clients have successfully downloaded all the blocks. Then, it just substitutes the value of DBin to DB to realise the abstract attempted substitution from AllDL.

As we introduce two new events, we have to show their convergence by introducing the variant given in Fig. 2.3. Each time Start1DL and Finish1DL are activated, their actions increase the numbers of incoming or finished in DBin, therefore the variant clearly decreases.

Figure 2.4 gives an extract of the operational semantics of this first refinement in terms of transition system, with N=2 and K=2. In this figure, a small dot in the matrix indicates that the block has not been downloaded yet, while an empty (resp. filled) bullet represents that the block is incoming (resp. successfully downloaded). For readability purposes, transitions in this TS have been annotated with the corresponding parameter values. The indicated v corresponds to the value of the variant in the corresponding state. As expected, the value of v decreases each time a new event is performed.

Remark that, at this point, any download attempt is ultimately successful. The purpose of the

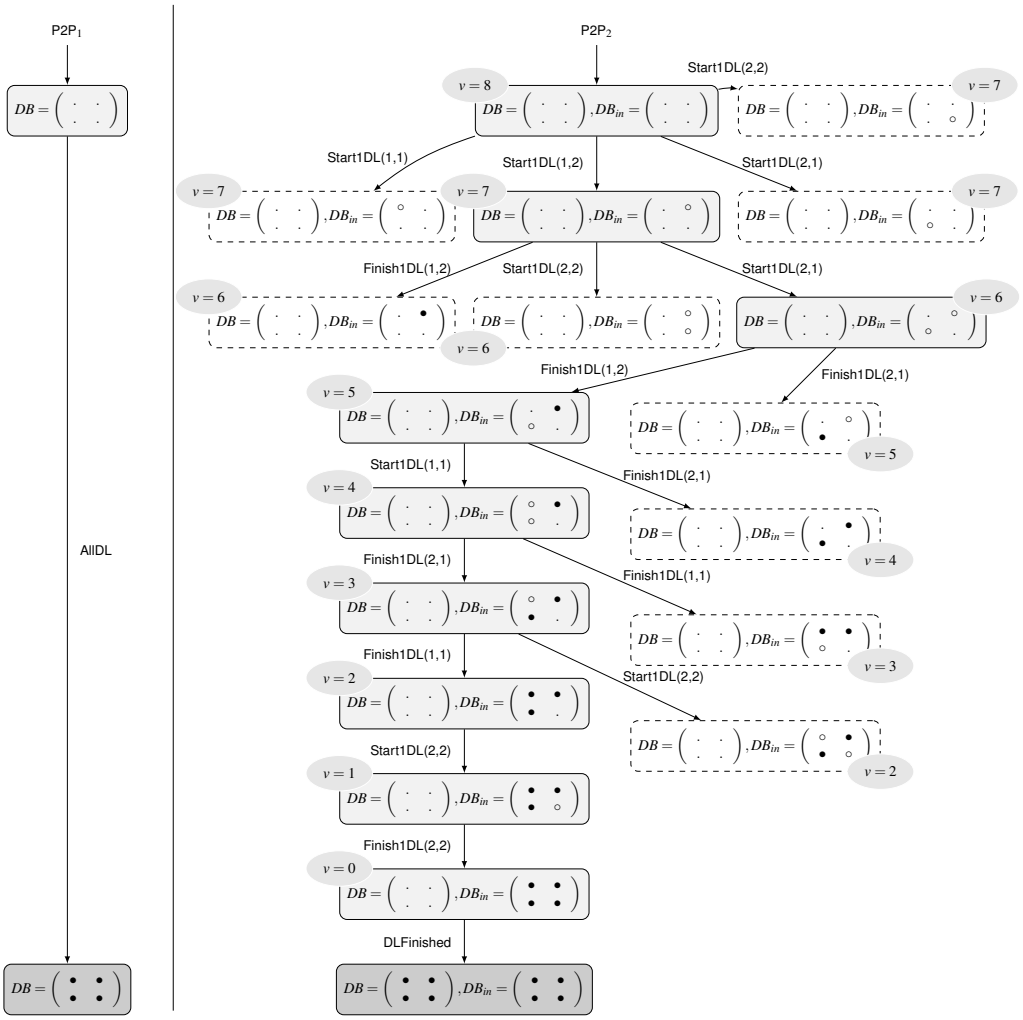


Figure 2.4 – Extract of the transition system of the first refinement of the simple P2P protocol, with N=2 and K=2

next refinement step is therefore to introduce failures in the block download process.

Introducing failures In this refinement $\mathsf{P2P}_3$, we want to take into consideration some possible failures during the download process. More precisely, two possible failures can occur when a download is incoming: a failure can be critical (in this case, the download must be aborted) or not (in this case, the download continues). We therefore simply add to the previous Event-B model a new event FailureDL, given in Fig 2.5. Its action non-deterministically chooses a value from {empty, incoming}: empty is chosen when the failure aborts the download and incoming is chosen otherwise. This new event has the same guard as the event FinishDB. As a consequence, both events are enabled at the same time and the choice between them is non-deterministic,

FailureDL $\widehat{=}$
any c, b **where**
c $\in$ 1..N $\wedge$ b $\in$ 1..K $\wedge$
DBin(c$\mapsto$b)=incoming
then
DBin(c$\mapsto$b):$\in${empty,incoming}
end

Figure 2.5 – New event introduced in the second refinement step

which models the uncontrolled occurrence of failures. An extract of the operational semantics of this new model is provided in Figure 2.6.

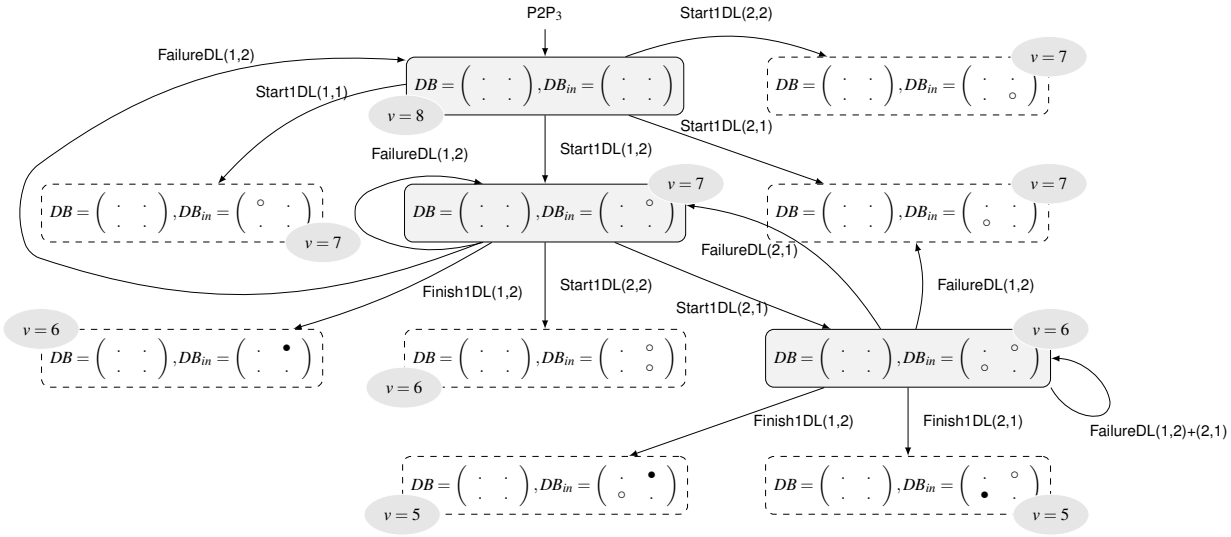


Figure 2.6 – Extract of the transition system of the second refinement of the simple P2P protocol, with N=2 and K=2

As we have introduced a new event, we need to prove its convergence. Unfortunately, due to the non-deterministic nature of the assignment in the event FailureDL, it is impossible to provide a variant that decreases regardless of its outcome. The refinement between this new model and the previous one is therefore not correct.

We will address the same problem in the probabilistic setting and prove that, in this case, the probabilistic version of FailureDL almost-certainly converges. As a consequence, unlike here, the refinement is correct in the probabilistic setting.

2.4 Fully probabilistic Event-B

We recall that our goal is to introduce probabilistic reasoning within Event-B. In this section, we present the basic elements of syntax and semantics for fully probabilistic Event-B models. We begin by presenting the sources of non-determinism in Event-B and explaining how they can be replaced by probabilistic choices in the context of fully probabilistic models. We then present the syntax of such fully probabilistic Event-B models. In order to ensure the consistency of these models, we present the set of new POs specific to the introduced elements and how standard POs can be adapted in this context. Finally, in order to prove the correctness of our approach, we propose operational semantics of such models in terms of Markov chains.

2.4.1 Introducing probabilistic choices

In Event-B, non-determinism can appear in three places: the choice of the enabled event to be executed, the choice of the parameter value to be taken and the choice of the value to be assigned to a given variable in a non-deterministic assignment. To obtain a *fully probabilistic Event-B model*, we propose to replace all these non-deterministic choices with probabilistic ones. In the following, we go through these three sources of non-determinism and explain how to turn them into probabilistic choices.

Choice of the enabled event In standard Event-B, when several events are enabled in a given configuration, the event to be executed is chosen non-deterministically. In order to resolve this non-deterministic choice, we propose to equip each probabilistic event with a *weight*. In configurations where several probabilistic events are enabled, the probability of choosing one of them will therefore be computed as the ratio of its weight against the total value of the weights of all enabled events in this state. Using weights instead of actual probability values is convenient as the set of enabled events evolves with the configuration of the system. Using probability values instead would require to normalise them in all configurations. Moreover, for the sake of expressivity, we propose to express the weight $\mathsf{W}_i(\bar{v})$ of a probabilistic event e_i as an expression over the variables $\bar{v}$ of the fully probabilistic Event-B model. The probability of executing a given event can therefore evolve as the system progresses. A probabilistic event is therefore allowed to be executed only if *i)* its guards is fulfilled and *ii)* its weight is strictly positive.

Choice of the parameter values In standard Event-B, events can be equipped with parameters. In each configuration where this is possible, a valuation of the parameters is chosen such that the

guard $\mathsf{G}_i(\bar{t},\bar{v})$ of the event is satisfied. When there are several such parameter valuations, one of them is selected non-deterministically. We therefore propose to replace this non-deterministic choice by a uniform choice over all parameter valuations ensuring that the guard of the event is satisfied. The uniform distribution is a default choice but our results can be extended to any other discrete distribution.

Non-deterministic assignments Recall that non-deterministic assignments in Event-B are expressed in two forms: predicate non-deterministic assignments and enumerated non-deterministic assignments.
We propose to replace predicate non-deterministic assignments by *predicate probabilistic assignments* written

$$\mathsf{x} :\oplus \mathsf{Q}_x(\bar{t},\bar{v},\mathsf{x}')$$

Instead of choosing non-deterministically among the values of x' such that the predicate $\mathsf{Q}_x(\bar{t},\bar{v},\mathsf{x}')$ is true as in standard predicate non-deterministic assignments, we propose to choose this new value using an uniform distribution. For simplicity reasons, we enforce that this uniform distribution must be discrete, and therefore that the set of values x' such that $\mathsf{Q}_x(\bar{t},\bar{v},\mathsf{x}')$ is true must always be finite. As above, the uniform distribution we propose by default could be replaced by any other discrete distribution.
We propose to replace enumerated non-deterministic assignments by *enumerated probabilistic assignments* written

$$\mathsf{x} := \mathsf{E}_1(\bar{t},\bar{v}) \mathbin{@} \mathsf{p}_1 \oplus \ldots \oplus \mathsf{E}_m(\bar{t},\bar{v}) \mathbin{@} \mathsf{p}_m$$

In this structure, the variable x is assigned the expression E_i with probability p_i [2]. In order to define a correct probability distribution, each p_i must be strictly positive and smaller or equal to 1, and they must sum up to 1. Although rational numbers are not natively handled in Event-B, we assume that the context of the model allows the use of rational numbers. In practice, that can be done by defining a "Rational" theory in Rodin using the theory plug-in providing capabilities to define and use mathematical extensions to the Event-B language and the proving infrastructure [BM13].
Remark that standard deterministic assignments are retained, but can also be considered as enumerated probabilistic assignments where $\mathsf{m}=1$.

2. In this version, we only consider constant values for the probabilities in enumerated probabilistic assignments, but this could be extended to expressions with a few adaptations.

$e_i \mathrel{\widehat{=}}$
weight $W_i(\bar{v})$
any $\bar{t}$ **where**
$G_i(\bar{t},\bar{v})$
then
$SP_i(\bar{t},\bar{v})$
end

Refining all non-deterministic choices into probabilistic choices has side effects on the syntax of events and models. In probabilistic Event-B, we therefore propose to use the syntax on the side for a probabilistic event e_i where $W_i(\bar{v})$ is the weight of the event, $G_i(\bar{t},\bar{v})$ is the guard of the event and $SP_i(\bar{t},\bar{v})$ is a *probabilistic action*, *i.e.*, an action consisting only of deterministic and *probabilistic* assignments which are executed in parallel. Remark that the Before-after predicate $SP_i(\bar{t},\bar{v},\bar{v}')$ of such a probabilistic event will be identical to the BA of its standard (non-deterministic) counterpart.
For simplicity reasons we impose, as in standard Event-B, that the initialisation event must be deterministic. The results we present in the rest of the chapter can nevertheless easily be extended to probabilistic initialisation events.

Definition 12 (Fully Probabilistic Event-B Model)**.** A Fully probabilistic Event-B model is a tuple $M=(\bar{v},I(\bar{v}),PEvts,\ Init\ )$ where $\bar{v}=\{v_1 \ldots v_n\}$ is a set of variables, $I(\bar{v})$ is the invariant, PEvts is a set of *probabilistic* events and Init is the initialisation event.

Running Example A probabilistic version of the P2P model from Section 2.3 is given in Fig. 2.7. This model has the same variables, the same invariants and the same events as the Event-B model from Figs. 2.3 and 2.5.
The events of the model $P2P_P$ are annotated with specific weights. The risk of download failures decreases with the number of successful downloads: each time a block is successfully downloaded, the weight of Finish1DL increases whereas the weight of FailureDL decreases. The weight of Start1DL models that the probability of starting a new download decreases with the number of blocks being currently downloaded.
In case of failure, we fix the probability of aborting the download to 40%. This probability is introduced in the event FailureDL by using an enumerated probabilistic assignment instead of a non-deterministic one: the variable $DBin(c\mapsto b)$ is assigned the value empty with a probability $\frac{4}{10}$ (the download aborts) and the value incoming with a probability $\frac{6}{10}$ (the download continues).

2.4.2 Consistency

As in standard Event-B, the consistency of a fully probabilistic Event-B model is defined by means of proof obligations (POs). In this section, we therefore introduce new POs specific to fully probabilistic Event-B and explain how we adapt standard Event-B POs in order to prove the consistency of fully probabilistic Event-B models.

MODEL
P2P$_P$

VARIABLES
DB
DBin

INVARIANTS
DB$\in 1..N \times 1..K \rightarrow \{empty,finished\} \wedge$
DBin $\in 1..N \times 1..K \rightarrow \{empty,incoming,finished\} \wedge$
$\forall$ c . (c $\in$ 1..N
$\Rightarrow$ card($\{b \mid b \in 1..K \wedge DBin(c \mapsto b)=incoming\}) \leq 1$)

VARIANT
$2 \times N*K$
$- 2 \times$ card($\{c \mapsto b \mid n \in 1..N \wedge b \in 1..K \wedge DBin(c \mapsto b) = finished\}$)
$-$ card($\{c \mapsto b \mid n \in 1..N \wedge b \in 1..K \wedge DBin(c \mapsto b) = incoming\}$)

EVENTS

Init $\widehat{=}$
begin
DB:=$(1..N \times 1..K) \times \{empty\} \parallel$
DBin:=$(1..N \times 1..K) \times \{empty\}$
end

DLFinished $\widehat{=}$
weight N*K
when
DB=$(1..N \times 1..K) \times \{empty\} \wedge$
DBin=$(1..N \times 1..K) \times \{finished\}$
then
DB:=DBin
end

Start1DL $\widehat{=}$
weight
N*K
$-$ card($\{c \mapsto b \mid n \in 1..N \wedge b \in 1..K \wedge DBin(c \mapsto b)=incoming\}$)
any c, b **where**
c $\in 1..N \wedge b \in 1..K \wedge$
$DBin(c \mapsto b)=empty \wedge$
card($\{k \mid k \in 1..K \wedge DBin(c \mapsto k)=incoming\}$)=0
then
$DBin(c \mapsto b):=incoming$
end

Finish1DL $\widehat{=}$
weight
card($\{c \mapsto b \mid n \in 1..N \wedge b \in 1..K \wedge DBin(c \mapsto b)=finished\}$) +1
any c, b **where**
c $\in 1..N \wedge b \in 1..K \wedge$
$DBin(c \mapsto b)=incoming$
then
$DBin(c \mapsto b):=finished$
end

FailureDL $\widehat{=}$
weight
N*K
$-$ card($\{c \mapsto b \mid n \in 1..N \wedge b \in 1..K \wedge DBin(c \mapsto b)=finished\}$)
any c, b **where**
c $\in 1..N \wedge b \in 1..K \wedge$
$DBin(c \mapsto b)=incoming$
then
$DBin(c \mapsto b):=empty$ @4/10 $\oplus$ incoming @6/10
end

END

Figure 2.7 – Probabilistic version of the simple P2P protocol

Specific POs for fully probabilistic Event-B

We start by presenting new POs specific to fully Probabilistic Event-B.

Numeric weight For simplicity reasons, we impose that the expression $W_i(\bar{v})$ representing the weight of a given probabilistic event must evaluate to natural numbers.

$I(\bar{v}) \wedge G_i(\bar{t},\bar{v}) \vdash W_i(\bar{v}) \in NAT$	(event/WGHT/NAT)

Parameter values finiteness In order to be able to use a discrete uniform distribution over the set of parameter valuations ensuring that the guard of a probabilistic event is satisfied, we impose that this set must be finite.

$I(\bar{v}) \vdash$ finite $(\{\bar{t} \mid G_i(\bar{t},\bar{v})\})$	(event/param/pWD)

Enumerated probabilistic assignments well-definedness and feasibility In all enumerated probabilistic assignments, it is necessary to ensure that the discrete probability values $\mathsf{p}_1 \ldots \mathsf{p}_n$ define a correct probability distribution. Formally, this leads to two POs:

(1) Probability values p_i in enumerated probabilistic assignments are strictly positive and smaller or equal to 1.

$\vdash\ 0 < \mathsf{p}_i \leq 1$ (event/assign/pWD1)

(2) The sum of the probability values $\mathsf{p}_1 \ldots \mathsf{p}_n$ in enumerated probabilistic assignments must be equal to 1.

$\vdash\ \mathsf{p}_1 + \ldots + \mathsf{p}_n = 1$ (event/assign/pWD2)

Feasibility of enumerated probabilistic assignments is trivial: as soon as at least one expression $\mathsf{E}_i(\bar{t},\bar{v})$ is present and well-defined, it always returns a value.

Predicate probabilistic assignment well-definedness and feasibility In order to define a discrete uniform distribution over the set of values of a variable x making the predicate $\mathsf{Q}_x(\bar{t},\bar{v},\mathsf{x}')$ of the corresponding assignment satisfied, we impose that this set must be finite.

$\mathsf{I}(\bar{v}) \wedge \mathsf{G}_i(\bar{t},\bar{v}) \wedge \mathsf{W}_i(\bar{v}) > 0 \vdash \text{finite}(\{\mathsf{x}' \mid \mathsf{Q}_x(\bar{t},\bar{v},\mathsf{x}')\})$ (event/assign/pWD3)

Feasibility of predicate probabilistic assignments is ensured by the standard feasibility PO [Abr10] inherited from Event-B. It ensures that the set $\{\mathsf{x}' \mid \mathsf{Q}_x(\bar{t},\bar{v},\mathsf{x}')\}$ is not empty.

Modifications to standard POs

In standard Event-B, if we want to prove that an event is enabled, we need to prove that its guard is satisfied. However, in fully probabilistic Event-B, we additionally need to prove that its weight is strictly positive. We therefore modify standard and optional Event-B POs as follows.

Invariant preservation The invariant must be preserved by all enabled probabilistic events.

$\mathsf{I}(\bar{v}) \wedge \mathsf{G}_i(\bar{t},\bar{v}) \wedge \mathbf{W}_i(\bar{v}) > \mathbf{0} \wedge \mathsf{SP}_i(\bar{t},\bar{v},\bar{v}') \vdash \mathsf{I}(\bar{v}')$ (event/pINV)

Deadlock freedom In all acceptable configurations, there must exist at least one enabled probabilistic event.

$\mathsf{I}(\bar{v}) \vdash (\mathsf{G}_1(\bar{t},\bar{v}) \wedge \mathbf{W}_1(\bar{v}) > \mathbf{0}) \vee \ldots \vee (\mathsf{G}_n(\bar{t},\bar{v}) \wedge \mathbf{W}_n(\bar{v}) > \mathbf{0})$ (model/pDLF)

For the sake of understanding, we hereby insist on the separation between the guard of an event, which reflects the classical notion of enabledness, and the fact that its weight must be strictly

positive. Obviously, one could also automatically re-write the guard of all probabilistic events in order to include the condition on its weight. This solution would allow conserving most of the standard Event-B consistency POs without modifications in the probabilistic setting.

Running example

Consider the fully probabilistic Event-B model $\mathsf{P2P}_P$ given in Fig. 2.7: all the weight expressions are natural numbers (event/WGHT/NAT) and, for each event, the number of acceptable parameter valuations is finite (event/param/pWD). For the probabilistic enumerated assignment on the event FailureDL, each given probability is a rational p such that $0 < p \leq 1$ (event/assign/pWD1) and their sum is clearly equal to 1 (event/assign/pWD2). The invariant is always preserved by each probabilistic version of the events (event/pINV). The model $\mathsf{P2P}_P$ is therefore *consistent*.

2.4.3 Semantics

As explained in Section 2.2.2, the operational semantics of standard Event-B models is expressed in terms of Labelled Transition Systems. In the following, we extend this work by presenting the operational semantics of fully probabilistic Event-B models in terms of Discrete Time Markov Chains (MC).
Remark that our goal, unlike in [TTL09, TTL15] is not to translate our models into MCs and use standard model checking techniques to verify them. Instead, we aim at reasoning directly on fully probabilistic Event-B models and benefiting from the symbolic proof mechanism that is the signature of the Event-B approach. The following MC semantics are nevertheless introduced as a demonstration of the correctness of our approach and results.

Notations

Let M=($\bar{v}$, I($\bar{v}$), PEvts, Init) be a fully probabilistic Event-B model and σ be a valuation of its variables. Given a variable $x \in \bar{v}$, we write $[\sigma]x$ for the value of x in σ. Given an expression $E(\bar{v})$ over variables in $\bar{v}$, we write $[\sigma]E(\bar{v})$ (or $[\sigma]E$ when clear from the context) for the evaluation of $E(\bar{v})$ in the context of σ. Given an expression $E(\bar{t},\bar{v})$ over variables and parameters, we write $[\sigma,\theta]E(\bar{t},\bar{v})$ for the evaluation of $E(\bar{t},\bar{v})$ under parameter valuation θ and variable valuation σ. Given a probabilistic event e_i with a set of parameters $\bar{t}$ and a valuation σ of the variables, we write $T_\sigma^{e_i}$ for the set of parameter valuations θ such that the guard of e_i evaluates to `true` in the context of σ and θ. Formally, $T_\sigma^{e_i} = \{\theta \mid [\sigma,\theta]G_i(\bar{t},\bar{v}) = true\}$. Recall that parameter valuations are chosen uniformly on this set. We write $P_{T_\sigma^{e_i}}$ for the uniform distribution on the set $T_\sigma^{e_i}$.

Given a valuation σ of the variables and a probabilistic event e_i, we say that e_i is *enabled* in σ iff (a) the weight of e_i evaluates to a strictly positive value in σ and (b) either e_i has no parameter and its guard evaluates to true in σ or there exists at least one parameter valuation θ such that the guard of e_i evaluates to true in the context of σ and θ, *i.e.*, $T_\sigma^{e_i} \neq \varnothing$.
Let e_i be a probabilistic event in PEvts and let x be a variable modified by e_i. Recall that x can be modified only by one assignment within the action of e_i. If x is modified by an enumerated probabilistic assignment $(x := E_1(\bar{t},\bar{v})@_{p_1} \oplus \ldots \oplus E_m(\bar{t},\bar{v})@_{p_m}\ (m \geq 1))$, then we write $\mathcal{E}_{e_i}(x)$ for the set of all expressions that can be assigned to the variable x by this assignment.

$$\mathcal{E}_{e_i}(x) = \{E_1(\bar{t},\bar{v}), ..., E_m(\bar{t},\bar{v})\}$$

The probability of choosing an expression E_i among all others expressions is written $P_x^{e_i}(E_i) = p_i$. Given a probabilistic event e_i, we write $Var(e_i)$ for the set of variables in $\bar{v}$ that are modified by the action of e_i, *i.e.*, the variables that appear on the left side of an assignment in $SP_i(\bar{t},\bar{v})$. Recall that a variable $x \in Var(e_i)$ must be on the left side of either a predicate probabilistic assignment or an enumerated probabilistic assignment. Let $e_i \in$ PEvts be a probabilistic event, $x \in Var(e_i)$ be a variable, σ, σ' two valuations of the variables $\bar{v}$ and θ a valuation of the parameter values associated to the event e_i such that e_i is enabled in the context of σ and θ, and leads the system to σ'.
If x is modified by an enumerated probabilistic assignment of e_i, then we write $\mathcal{E}_{e_i}(x)|_{\sigma,\theta}^{\sigma'}$ for the set of expressions in $\mathcal{E}_{e_i}(x)$ such that their evaluation in the context of σ and θ returns the value of x in the valuation σ'.
Formally,

$$\mathcal{E}_{e_i}(x)|_{\sigma,\theta}^{\sigma'} = \{E \in \mathcal{E}_{e_i}(x) \mid [\sigma,\theta](E(\bar{t},\bar{v})) = [\sigma']x\}$$

If e_i is not equipped with parameters, then this subset is written $\mathcal{E}_{e_i}(x)|_{\sigma}^{\sigma'}$.
If x is modified by a predicate probabilistic assignment $(x : \oplus Q_x(\bar{t},\bar{v},x'))$, then we write $\mathcal{V}_{\theta,\sigma}^{e_i}(x)$ for the set of values x' that make the predicate $Q_x(\bar{t},\bar{v},x')$ true when evaluated in σ and θ.

$$\mathcal{V}_{\theta,\sigma}^{e_i}(x) = \{x' \mid [\sigma,\theta]Q_x(\bar{t},\bar{v},x') = true\}$$

If e_i is not equipped with parameters, then this subset is written $\mathcal{V}_{\sigma}^{e_i}(x)$.
Let e_i be a probabilistic event and let x be a variable in $Var(e_i)$, given an original valuation σ of the variables, a valuation θ of the parameters of e_i and a target valuation σ' of the variables, we write $P_{\sigma,\theta}^{e_i}(x,\sigma')$ for the probability that x is assigned the new value $[\sigma']x$ when executing e_i from the valuation σ and with parameter valuation θ. If e_i is not equipped with parameters, this

is written $P^{e_i}_{\sigma}(x,\sigma')$. In the following, we always use the more general notation and assume that it is replaced with the specific one when there are no parameters. Formally, this probability is given by:

(1) if x is modified by an enumerated probabilistic assignment, then:

$$P^{e_i}_{\sigma,\theta}(x,\sigma') = \sum_{E \in \mathcal{E}_{e_i}(x)|^{\sigma'}_{\sigma,\theta}} P^{e_i}_{x}(E)$$

(2) if x is modified by a predicate probabilistic assignment, then:

$$P^{e_i}_{\sigma,\theta}(x,\sigma') = \frac{1}{card(\mathcal{V}^{e_i}_{\theta,\sigma}(x))} \text{ if } [\sigma']x \in \mathcal{V}^{e_i}_{\theta,\sigma}(x) \text{ and 0 otherwise.}$$

MC operational semantics

Informally, the operational semantics of a fully probabilistic Event-B model M=($\bar{v}$, I($\bar{v}$), PEvts, Init) is a Probabilistic LTS $[\![M]\!] = (S, \mathsf{Acts}, P, s_0, AP, L)$ where the states, labels, actions, atomic propositions and initial state are similarly obtained as for the standard LTS semantics of Event-B. The only difference with the standard LTS semantics is that the transitions are equipped with probabilities, which we explain below. In the following, we identify the states with the valuations of the variables defined in their labels.
Intuitively, the transition probabilities are obtained as follows: Let $e_i \in \mathsf{PEvts}$ be a probabilistic event, $x \in \bar{v}$ be a variable and s, s' be two states of $[\![M]\!]$ such that (s, e_i, s') is a transition in the standard LTS semantics, *i.e.*, where e_i is enabled in s and there exists a parameter valuation $\theta \in T_s^{e_i}$, if any, such that the action of e_i may take the system from s to s'. The probability assigned to transition (s, e_i, s') is then equal to the product of ***(1)*** the probability that the event e_i is chosen from the set of enabled events in state s, ***(2)*** the probability of choosing each parameter valuation θ, and ***(3)*** the overall probability that each modified variable is assigned the value given in s' under parameter valuation θ.

Definition 13 (Fully Probabilistic Event-B operational Semantics)**.** The operational semantics of a fully probabilistic Event-B model M=($\bar{v}$, I($\bar{v}$), PEvts, Init) is a *PLTS* $[\![M]\!] = (S, \mathsf{Acts}, P, s_0, AP, L)$ where $S, \mathsf{Acts}, s_0, AP$, and L are defined as in the standard LTS semantics of Event-B models (see Section 2.2.2), and $P : S \times \mathsf{Acts} \times S \to [0,1]$ is the transition probability function such that for a given state s, for all $e_i, s' \in \mathsf{Acts} \times S$, we have $P(s, e_i, s') = 0$ if $e_i \notin \mathsf{Acts}(s)$ or $\exists x \in X \backslash \{Var(e_i)\}$ st $[s]x \neq [s']x$ and otherwise

$$P(s,e_i,s') = \underbrace{\frac{[s]W_i(\bar{v})}{\sum_{e_j \in \mathsf{Acts}(s)}[s]W_j(\bar{v})}}_{(1)} \times \sum_{\theta \in T_s^{e_i}} \Big(\underbrace{P_{T_s^{e_i}}(\theta)}_{(2)} \times \underbrace{\prod_{x \in Var(e_i)} P_{s,\theta}^{e_i}(x,s')}_{(3)} \Big)$$

As expected, the following proposition shows that the semantics of a fully probabilistic Event-B model as defined above is indeed a MC. The proof is available in [ADL19b].

Proposition 7. *The operational semantics of a fully probabilistic Event-B model M satisfying the POs given in Section 2.4.2 is a MC.*

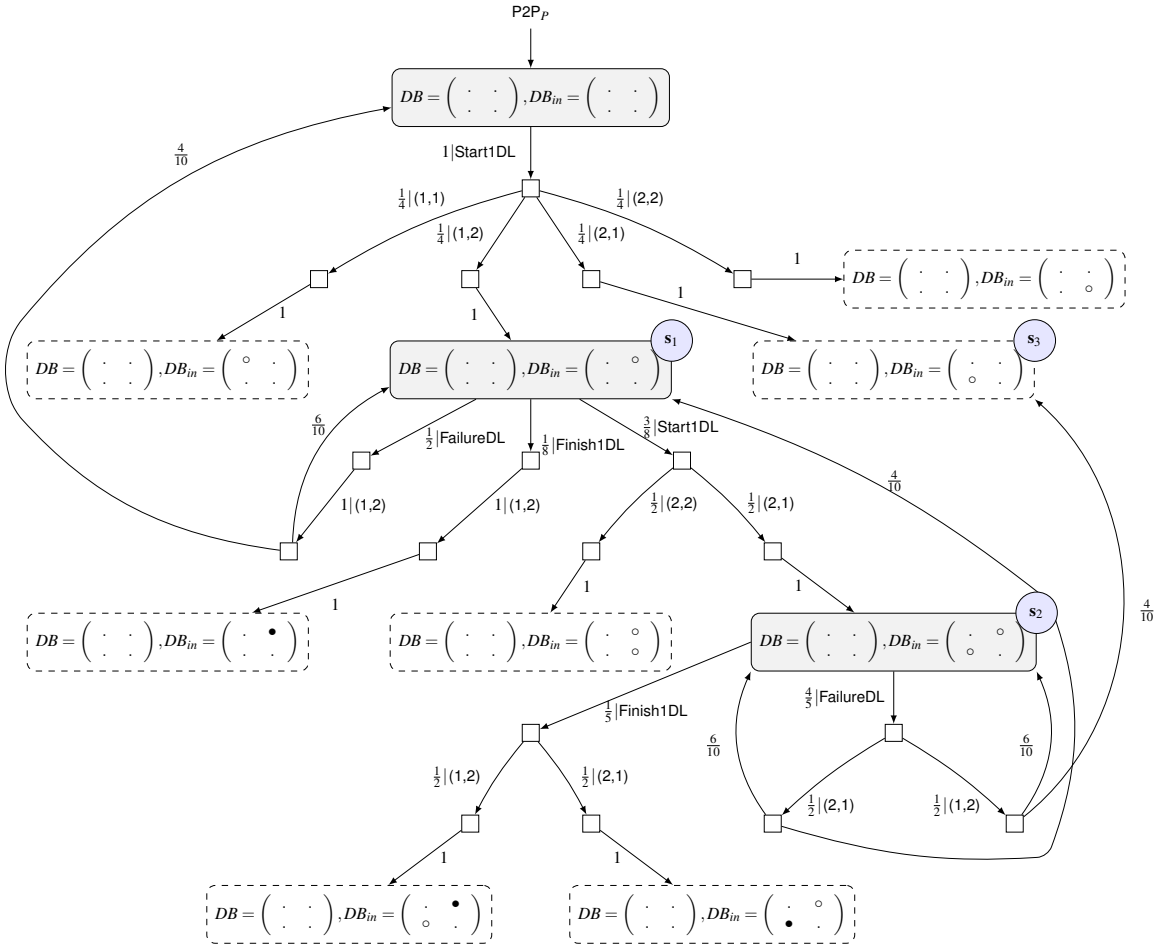


Figure 2.8 – Extract of the Detailed construction of the MC of the simple P2P protocol, with N=2 and K=2

Running example

Fig. 2.8 presents the first steps of the detailed construction of the MC corresponding to the fully probabilistic Event-B model given in Fig. 2.7, with the number of clients N and the numbers of blocks K fixed to 2. We only present this detailed construction to illustrate the operational semantics of our model as defined above. This MC will not be used within the design process in probabilistic Event-B.

We now explain how some of the probability values in the MC from Fig. 2.8 are computed. We only focus on interesting (and complex) examples and leave out the rest of the computation to the reader. From the state referenced $(\mathbf{s_1})$ on Fig. 2.8, three events are enabled:

— FailureDL with a weight value of 4. The probability of choosing this event is therefore $\frac{4}{4+3+1} = \frac{1}{2}$;

— Start1DL with a weight value of 3. The probability of choosing this event is therefore $\frac{3}{4+3+1} = \frac{3}{8}$;

— Finish1DL with a weight value of 1. The probability of choosing this event is therefore $\frac{1}{4+3+1} = \frac{1}{8}$.

When choosing the event Finish1DL, only one valuation for the parameters is possible with a probability of 1. The corresponding action is deterministic, and therefore executed with probability 1. The global probability of leaving the state $(\mathbf{s_1})$ using the event Finish1DL is therefore $\frac{1}{8} \times 1 \times 1 = \frac{1}{8}$.

When choosing the event Start1DL, two possible valuations for the parameters are possible, with a probability $\frac{1}{2}$ for each of them. The action corresponding to event Start1DL is deterministic, and the parameter valuation $(2,1)$ allows to reach state $(\mathbf{s_2})$. As a consequence, the global probability of reaching $(\mathbf{s_2})$ from $(\mathbf{s_1})$ using the event Start1DL is $\frac{3}{8} \times \frac{1}{2} \times 1 = \frac{3}{16}$.

When choosing the event FailureDL, only one valuation for the parameters is possible. The corresponding action is probabilistic, leading to two different states (with probabilities $\frac{6}{10}$ of going back to $(\mathbf{s_1})$ and $\frac{6}{10}$ of going to another state). As a consequence, the global probability of looping on $(\mathbf{s_1})$ using the event FailureDL is $\frac{1}{2} \times 1 \times \frac{6}{10} = \frac{3}{10}$.

From the state referenced as $(\mathbf{s_2})$ on Fig. 2.8, we only focus on one interesting transition. Among the two events that can be enabled, we consider the event FailureDL: the probability of choosing this event is $\frac{4}{5}$. Two possible valuations for the parameters are then possible, with a probability of $\frac{1}{2}$ for each of them. Then, for each parameter valuation, the corresponding action is probabilistic, leading to different states. What makes this transition interesting is that for different parameter valuations, some actions lead to the same state:

— the global probability of returning to $(\mathbf{s_1})$ is $\frac{4}{5} \times \frac{1}{2} \times \frac{4}{10} = \frac{4}{25}$,

— the global probability of reaching $(\mathbf{s_3})$ is $\frac{4}{5} \times \frac{1}{2} \times \frac{4}{10} = \frac{4}{25}$,

— the global probability of looping on $(\mathbf{s_2})$ is $\frac{4}{5} \times (\underbrace{\frac{1}{2} \times \frac{6}{10}}_{(2,1)} + \underbrace{\frac{1}{2} \times \frac{6}{10}}_{(1,2)}) = \frac{12}{25}$, where $(2,1)$ and $(1,2)$ are the parameter valuations leading to these probabilistic choices.

After reduction (removal of the intermediate $\square$ states), we obtain the MC given Fig.2.9.

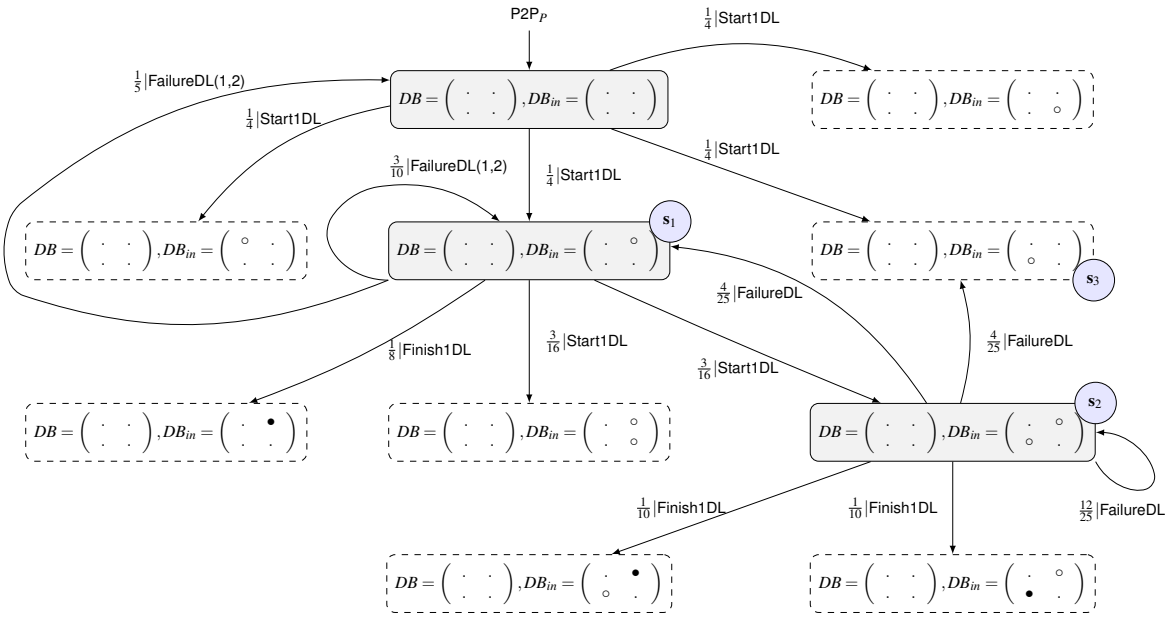


Figure 2.9 – Extract of the MC of the simple P2P protocol, with N=2 and K=2

2.5 Introducing probabilities in event-B models through refinement

Our main goal is to enable modeling probabilistic behaviours within Event-B. As explained earlier (and illustrated in Fig. 2.1), this can be done in several ways while preserving the refinement-based approach inherent to Event-B. The first way is to transform a standard (group of) event(s) into a probabilistic (group of) event(s) in a refinement step. This transformation therefore allows to turn a standard or mixed Event-B model into a mixed or fully probabilistic Event-B models. This transformation is straightforward but subject to certain conditions that we call *probabilisation feasibility* POs. These conditions ensures that the resulting model will

be consistent, *e.g.*, that the set of parameter values such that the guard of an event subject to probabilisation is finite. The study of this transformation has been left out of this document for the sake of conciseness, but can be found in [ADL19b].

A second way of introducing probabilistic information in the design process is by introducing new probabilistic events through (probabilistic) refinement. One principal aspect of refinement in Event-B is the addition within a refinement step of new variables and new events acting on those variables. In this section, we explain how to introduce new probabilistic events in a given (abstract) model. Regardless of the type of model (non-deterministic, mixed or fully probabilistic), it is necessary to show that the introduction of these new events cannot prevent the system from behaving as specified in the abstract model. Recall that this is usually done by proving that the set of new events introduced in the refinement step converges, *i.e.*, that events from this set cannot keep control indefinitely. As a consequence, at some point, the system must stop the execution of new events in order to execute the behaviour proposed in the abstract model.

We therefore propose a solution in order to prove that a given set of probabilistic events almost-certainly converges. In the following, we only focus on fully probabilistic Event-B models, but all the presented results can be generalised to standard or mixed Event-B models (see [ADL19b] for details).

In standard Event-B refinement, it is required to show that a given set of events *always converges*. On the contrary, in probabilistic Event-B, it is only required to prove that a given set of probabilistic events *almost-certainly* converges. In other words, we are interested in showing that, in all states of the system where convergent probabilistic events can be executed, the probability of eventually taking a non-convergent event or reaching a deadlock is 1 (*i.e.*, the probability of infinitely executing convergent events is 0).

This property has already been investigated in [HH07] and [Hoa14], in the context of events having probabilistic actions but where non-determinism is still present between events. In this context, Hallerstede and Hoang propose in [HH07] sufficient conditions for a set of events to almost-certainly converge. These conditions can be summarised as follows: As in standard Event-B, one needs to exhibit a natural number expression $\mathsf{V}(\bar{v})$ called a variant. Unlike in the standard setting, only one resulting valuation of the execution of *each* convergent event needs to decrease this variant. Indeed, in this case, the probability of decreasing the variant is strictly positive. Unfortunately, using such a permissive condition is not sufficient in our context: there might also be a strictly positive probability of increasing the variant. Therefore, Hallerstede and Hoang require the introduction of another natural number expression $\mathsf{U}(\bar{v})$ which must maximise the variant $\mathsf{V}(\bar{v})$ and never increase. The proposition from [HH07] is refined in [Hoa14], where

Hoang requires in addition that the probabilities considered in probabilistic assignments are bounded away from 0. This is ensured by requiring that the set of values that can be returned by a probabilistic assignment is finite.

Adaptation to probabilistic events

We now show how to adapt the results proposed in [HH07] and [Hoa14] to new probabilistic events introduced in a fully probabilistic Event-B model. Since there are no non-deterministic choices between enabled events, it is not anymore necessary to require that *all* enabled events in a given configuration may decrease the variant. We therefore start by relaxing the condition proposed in [HH07]: we only require that, in all configurations where a convergent event is enabled, there is *at least one* convergent event for which at least one resulting valuation decreases the variant.

(1) **Almost-certain convergence.** In all configurations where at least one convergent event is enabled, there must exist at least one valuation $\bar{v}$' obtained after the execution of one of these enabled events which decreases the variant.

$$\begin{array}{l} \mathsf{I}(\bar{v}) \ \wedge \ ((\mathsf{G}_i(\bar{t},\bar{v}) \ \wedge \mathsf{W}_i(\bar{v}) > 0) \vee \ldots \vee (\mathsf{G}_n(\bar{t},\bar{v}) \ \wedge \mathsf{W}_n(\bar{v}) > 0)) \vdash \\ \qquad (\exists\, \bar{v}'.\ \mathsf{G}_i(\bar{t},\bar{v}) \ \wedge \mathsf{W}_i(\bar{v}) > 0 \wedge \mathsf{SP}_i(\bar{t},\bar{v},\bar{v}') \wedge \mathsf{V}(\bar{v}') < \mathsf{V}(\bar{v})) \vee \ldots \vee \\ \qquad\quad (\exists\, \bar{v}'.\ \mathsf{G}_n(\bar{t},\bar{v}) \ \wedge \mathsf{W}_n(\bar{v}) > 0 \wedge \mathsf{SP}_n(\bar{t},\bar{v},\bar{v}') \wedge \mathsf{V}(\bar{v}') < \mathsf{V}(\bar{v})) \end{array} \qquad \text{(model/pVar)}$$

As in [HH07], we also require that convergent events can only be enabled when the variant is positive and that the variant is bounded above. In order to simplify the reasoning, we propose to use a constant bound U, as in [Hoa14].

(2) **Numeric variant.** Convergent events can only be enabled when the variant is greater or equal to 0.

$$\mathsf{I}(\bar{v}) \ \wedge \mathsf{G}_i(\bar{t},\bar{v}) \ \wedge \mathbf{W}_i(\bar{v}) > \mathbf{0} \ \vdash \ \mathsf{V}(\bar{v}) \in \mathsf{NAT} \qquad \text{(event/var/pNAT)}$$

(3) **Bounded variant.** Convergent events can only be enabled when the variant is less or equal to U.

$$\mathsf{I}(\bar{v}) \ \wedge \mathsf{G}_i(\bar{t},\bar{v}) \ \wedge \mathbf{W}_i(\bar{v}) > \mathbf{0} \ \vdash \ \mathsf{V}(\bar{v}) \leq \mathsf{U} \qquad \text{(event/pBOUND)}$$

Finally, the finiteness of the set of values that can be returned by a probabilistic assignment is already ensured by the syntax for enumerated probabilistic assignments and by PO (event/assign/pWD3) for predicate probabilistic assignments and their non-emptyness is ensured by the standard feasibility POs.

Inadequacy of adapted POs Unfortunately, as we deal with potentially infinite-state systems, POs 1–3 presented above are not anymore sufficient for proving that the probability of eventually executing a non-convergent event or reaching a deadlock is 1. Indeed, although the probability of decreasing the variant is always strictly positive because of PO (model/pVar) and although the number of values that can be returned by a given probabilistic assignment is always finite, the combination of event weights and parameter choice can make this value infinitely small in some cases. In this case, it is well known that almost-certain reachability/convergence is not ensured. This problem is a direct consequence of the unboundedness of the weights of convergent events as well as of the number of acceptable parameter values, which, by getting arbitrarily big, cause the probability of decreasing the variant to get arbitrarily small. Two examples illustrating this fact are given below.

Example 10 (Necessity of bounding event weights)**.** In this example, we show by means of an example of a probabilistic Event-B model the necessity of bounding the weights of new probabilistic events in order to ensure almost-certain convergence.
Consider the probabilistic Event-B model M1 and the corresponding MC semantics given in Fig. 2.10. This model has two variables: x and y and three events evt1, evt2 and evt3, two of which (evt1 and evt2) are convergent. The variant of this model is x and the bound on the variant is clearly U = 2.
In states where $x = 1$, only convergent events evt1 and evt2 are enabled and the local probability of choosing evt1 is $\frac{1}{y}$ while the local probability of choosing evt2 is $\frac{y-1}{y}$. In states where $x = 2$, only evt1 can be chosen with probability 1. In states where $x = 0$, the only enabled event is the (non-convergent) event evt3.
Clearly, the model M1 satisfies proof obligations (model/pVar), (event/var/pNAT) and (event/pBOUND). However, as we show below, the probability of eventually taking a non-convergent event is strictly smaller than 1 from all states where $x > 0$ because the probability of decreasing the variant, although strictly positive in all states, gets infinitely small from states where $x = 1$ as y increases. Without loss of generality, we compute the probability of eventually taking evt3 from the initial state where $x = 1$ and $y = 2$. The reasoning starting from other states is similar. This probability is equal to the sum of

(1) the probability of directly taking evt1 from $(1,2)$,

(2) the probability of reaching $(1,4)$ and taking evt1 from $(1,4)$,

(3) the probability of reaching $(1,8)$ and taking evt1 from $(1,8), \ldots$

Clearly, (1) is equal to $\frac{1}{2}$, (2) is equal to $\frac{1}{2} \cdot \frac{1}{4} = \frac{1}{8}$, (3) is equal to $\frac{1}{2} \cdot \frac{3}{4} \cdot \frac{1}{8} < \frac{1}{16}$ and in general, the

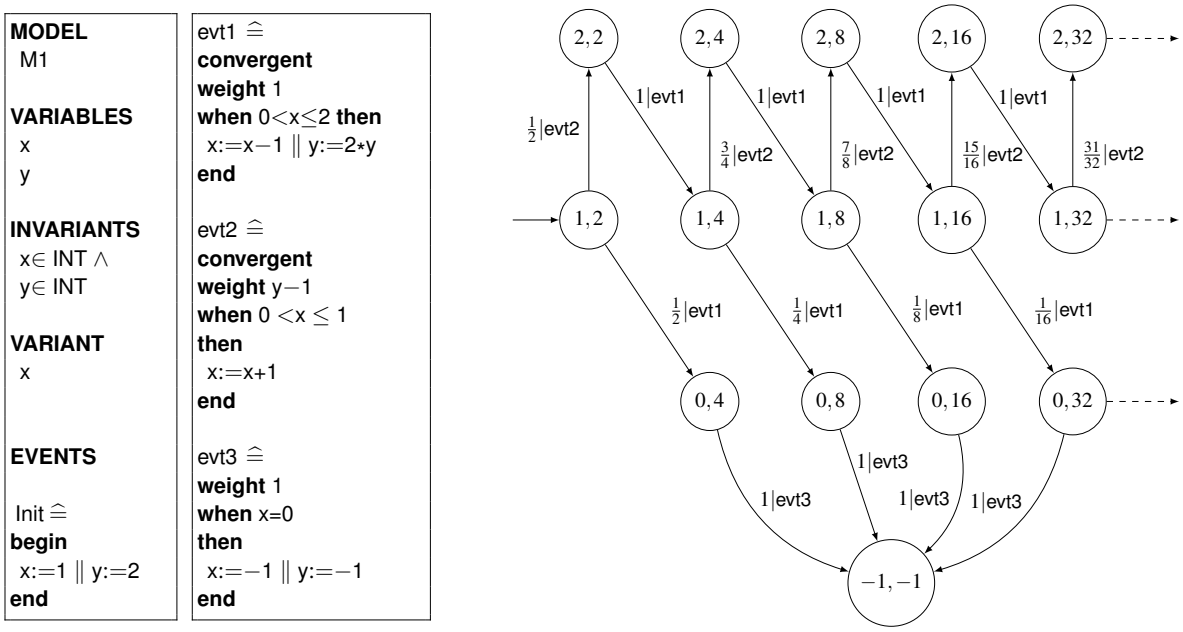


Figure 2.10 – Probabilistic Event-B model and MC semantics illustrating the necessity of bounding event weights to ensure almost-certain convergence

probability of reaching state $(1,2^i)$ with $i > 2$ and taking evt1 from this state is strictly smaller than $\frac{1}{2^{i+1}}$.

As a consequence, the probability of eventually taking evt3 from the initial state is strictly smaller than

$$\frac{1}{2} + \sum_{i=2}^{\infty} \frac{1}{2^{i+1}} = \frac{3}{4}$$

Therefore, M1 does not almost-certainly converge.

The behaviour we expose here is a direct consequence of the unboundedness of the weights of convergent events, which, by getting arbitrarily big, cause the probability of decreasing the variant to get arbitrarily small.

Example 11 (Necessity of bounding event parameter values)**.** We now use a similar example to show the necessity of bounding the number of admissible parameter values in new probabilistic events in order to prove their almost-certain convergence. The probabilistic Event-B model M2 and its corresponding MC semantics, given in Fig. 2.11 are similar to the ones presented in Fig. 2.10.

In this case also, we observe that the probability of eventually executing non-convergent event

MODEL
M2

VARIABLES
x
y

INVARIANTS
x∈ INT ∧
y∈ INT

VARIANT
x

EVENTS

Init ≙
begin
x:=1
‖ y:=1
end

evt1 ≙
convergent
weight 1
any t **where**
t ∈ {1.. 2^y} ∧ 0<x≤1
then
x:⊕((t=1 ∧ x'−x+t=0) or
(2≤t≤2^y ∧ x'−x−1=0))
‖ y:=y+1
end

evt2 ≙
convergent
weight 1
when x=2
then
x:=x−1
end

evt3 ≙
weight 1
when x=0
then
x:=−1 ‖ y:=−1
end

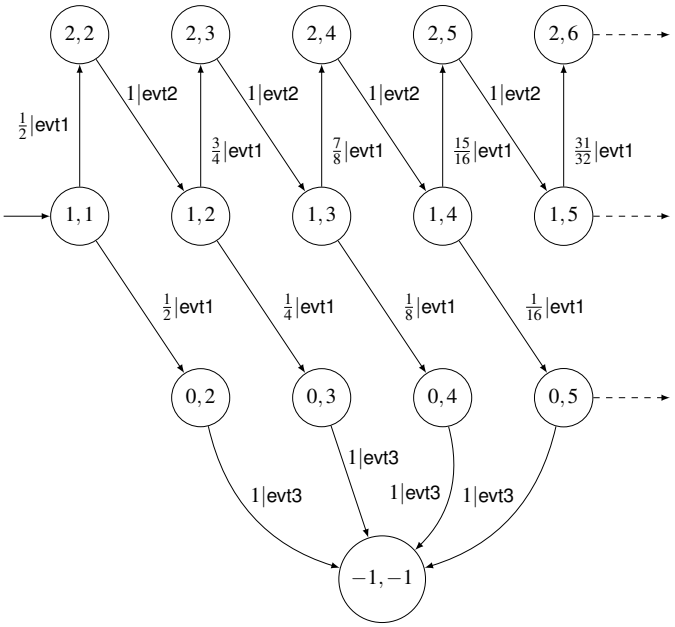


Figure 2.11 – Probabilistic Event-B model and MC semantics illustrating the necessity of bounding event parameter values to ensure almost-certain convergence

evt3 from the initial state is strictly smaller than $3/4$. The main difference is that, in M2, only the choice of parameter values is responsible for infinitely decreasing the probabilities of decreasing the variant.

Additional Proof Obligations. We therefore adapt classical results from infinite-state MC to our setting and propose sufficient conditions in terms of proof obligations to prove the almost-certain convergence of the set of new introduced events. Informally, the following POs ensure that the probability of decreasing the variant cannot get infinitely small by requiring that both the weights of convergent events and the number of potential values given to parameters in convergent events are bounded.

(4) **Bounded weight.** The weight of all convergent events must be bounded above by a constant upper bound BW.

$I(\bar{v}) \wedge G_i(\bar{t},\bar{v}) \vdash W_i(\bar{v}) \leq BW$	(event/wght/BOUND)

(5) **Bounded parameters.** The number of potential values for parameters in convergent events must be bounded above by a constant upper bound BP.

$I(\bar{v}) \vdash \mathsf{card}(\{\bar{t} \mid G_i(\bar{t},\bar{v})\}) \leq BP$	(event/param/BOUND)

The following theorem shows that the conditions presented above are sufficient for guaranteeing the almost-certain convergence of a given set of events in a probabilistic Event-B model.

Theorem 3. *Let M=($\bar{v}$, I($\bar{v}$), V($\bar{v}$), PEvts, Init) be a probabilistic Event-B model and* $PEvts_c \subseteq PEvts$ *a set of convergent events. If M satisfies the above POs (1-5), then the set* $PEvts_c$ *almost-certainly converges.*

The full proof of this theorem is particularly intricate and therefore left out of this document, but a detailed proof can be found in [ADL19b]. In the following, we provide the intuition of the proof.

Proof Sketch

Almost certain convergence of the Event-B model is proven using its operational semantics, i.e.*, showing the following* convergence property*: the probability measure of the set of runs that eventually reach a deadlock or use a non-convergent event is* 1*. In order to take into account the difference between convergent and non-convergent events, we start by defining a slightly extended version of the MC semantics of M. In this version, all the states are replicated in order to "remember" the last event executed.*

We observe that the MC semantics of M has a potentially infinite set of states. Therefore, showing the convergence property *is not trivial. In order to prove it, we use the* global coarseness *property introduced in [MHA07], which is a sufficient condition for the "decisiveness" of infinite-state Markov Chains. Formally, given a Markov Chain* $\mathcal{M} = (\mathcal{S}, s_0, \mathcal{P})$ *and a target set of states* $\mathcal{F} \subseteq \mathcal{S}$*, we say that* $\mathcal{M}$ *is globally coarse w.r.t.* $\mathcal{F}$ *iff there exists some minimal bound* $\alpha > 0$ *such that for all state* $s \in \mathcal{S}$*, the probability of eventually reaching* $\mathcal{F}$ *from s is either* 0 *or greater or equal to* α*. It is then shown in [MHA07] that whenever a Markov Chain* $\mathcal{M}$ *is globally coarse w.r.t. the set* $\mathcal{F}$*, the probability of eventually reaching either* $\mathcal{F}$ *or a set of states* $\widetilde{\mathcal{F}}$ *from which* $\mathcal{F}$ *cannot be reached is 1 from any state of* $\mathcal{M}$*.*

We therefore use this result on the semantics of M using a "clever" partition of the state space. □

Remark. The additionnal POs imposing boundedness of weights and event parameters are therefore sufficient (in addition to the adaptation of standard convergence POs presented earlier) for proving the convergence of a given set of events. While these POs are certainly restrictive, they are easily provable and seem consistent with the requirement on the boundedness of the variant. Identifying less restrictive conditions in general is still an open question.

Running example

Recall that, as explained at the end of Sec. 2.3, we cannot prove that the peer-to-peer protocol (with failures) always terminates, because we cannot prove the convergence of the non-deterministic Event-B model $\mathsf{P2P}_3$. Indeed, the event FailureDL fails to decrease the corresponding variant, therefore preventing convergence.
Now consider the fully probabilistic Event-B model $\mathsf{P2P}_P$ given in Fig. 2.7 from Sec. 2.4. To show that the protocol always terminates we have to show that the set of events {Start1DL, Finish1DL, FailureDL} almost-certainly converges (*i.e.*, converges with probability 1).

— The variant is numeric (event/var/pNAT) and we can take the expression $2 \times N \times K$ as a possible upper bound for the variant (event/pBOUND);

— The weights of the convergent events are bounded by $N \times K + 1$ (event/wght/BOUND);

— The possible parameter values are bounded by $N \times K$ (event/param/BOUND);

— Finally, in each possible configuration, the event Finish1DL decreases the variant with a positive probability (model/pVar).

Since the model $\mathsf{P2P}_P$ satisfies all the required POs, Theorem 3 ensures that the set {Start1DL, Finish1DL, FailureDL} almost-certainly converges.

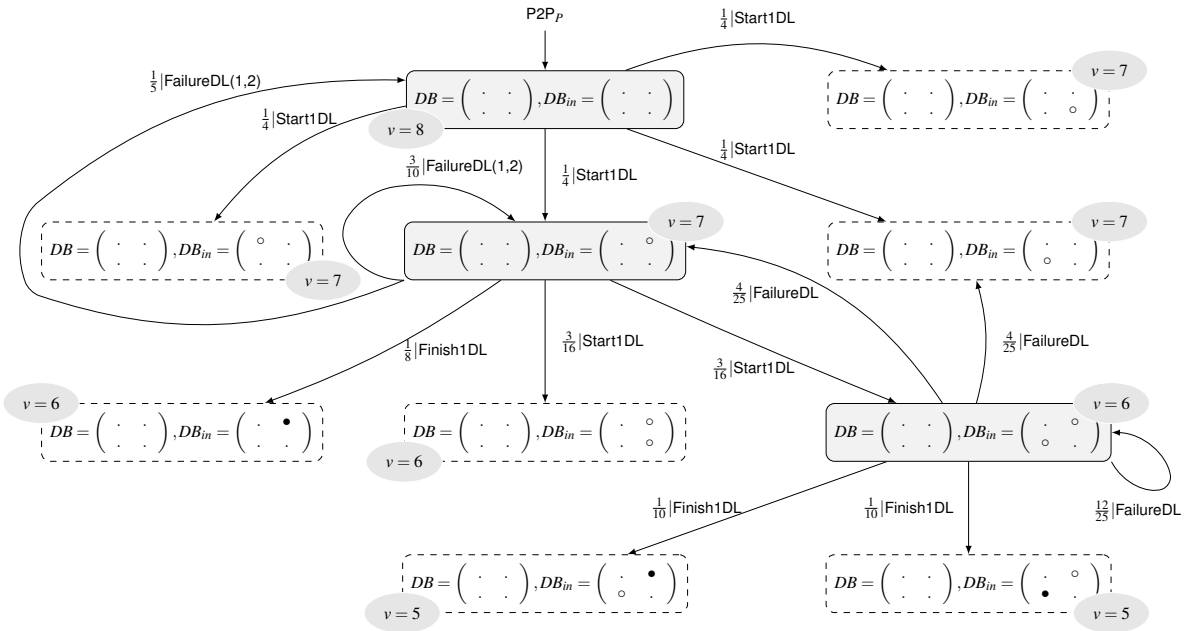


Figure 2.12 – Extract of the MC of the simple P2P protocol, with N=2 and K=2, illustrating the decrease of the variant

In order to illustrate that the variant indeed decreases with a positive probability from all states using probabilistic event Finish1DL, we provide in Fig. 2.12 an extract of the MC semantics of $\mathsf{P2P}_P$ (for N=2 and K=2), where all states are labelled with the value of the variant.

2.6 Conclusion

In this chapter, we have presented an extension to the Event-B formalism that allows describing models with probabilistic aspects. We have focused on fully probabilistic models but the complete theory including also mixed Event-B models has been developped and can be found in [ADL19b]. In this context, we have provided proof obligations for the consistency of fully probabilistic models and expressed their operational semantics in terms of probabilistic transition systems. Moreover, we have also explained how the addition of probabilistic information can be done either as a standalone artefact (probabilisation of a standard model) or as a part of the design process that can be interleaved with standard refinement steps. In particular, we have focused on the addition of new probabilistic events in fully probabilistic models and developed sufficient conditions in terms of proof obligations in order to show that a given set of probabilistic events is almost-certainly convergent, which is a required property in this context in standard Event-B. Most of our results have been implemented in a probabilistic plugin for the Rodin Platform. The Rodin tool [ABH+10] is an Eclipse-based IDE for designing models in Event-B. It allows the creation of Event-B models, the automatic generation of POs and it incorporates some provers for discharging the necessary POs. Rodin is based on a set of plugins, that facilitate its extension to support new functionalities.

Our plugin is still under development, but it already supports the specification of fully probabilistic Event-B models and the generation of some dedicated POs presented in this chapter. The plugin also allows the *probabilisation* of a non-deterministic Event-B model: it automatically generates the corresponding probabilistic Event-B model. Once this latter is generated, the developer must complete the weight of each probabilistic event and probability values for each quantitative probabilistic assignment.

2.7 Perspectives

This line of work has been dormant since the end of Mohamed Amine Aouadhi's thesis. In particular, our implementation of the probabilistic Even-B plugin for Rodin has been left at a

standstill and is still missing some essential features. We plan on taking over this implementation in the future.
On the theoretical side, although we have considered the addition of new probabilistic events in a probabilistic model, a complete counterpart to standard refinement for the probabilistic setting still eludes us. The problem mainly lies in two operations that are allowed in standard Event-B refinement: the split and merge operations.

— The split operation allows, in one refinement step, to transform one abstract event into multiple concrete events, while allowing the guards of these concrete events to be more restrictive than the guard of the abstract event. In the probabilistic setting, the problem mainly lies in the repartition of the weights of concrete events w.r.t the weight of the abstract event depending on which of the guards are satisfied. We have not found yet a satisfying solution that does not restrict in a too strict way the original split operation. Indeed, a simple but restrictive solution is to impose that the guards of the concrete events must be identical to the guard of the original abstract event. In this case, we only have to impose that the sum of the weights of the concrete events is equal to the weight of the abstract event.

— The merge operation, on the contrary, allows for a single concrete event to refine several abstract events at once. The same problem regarding the repartition of the weights of the original events arises in this setting, which has prevented us from finding a satisfying solution yet.

The verification of probabilistic properties has also been left aside for the moment, as their expression and meaning in the Event-B formalism is not trivial. The only way of proving probabilistic properties on a probabilistic Event-B model at the moment is to use its MC/MDP operational semantics, which is far from ideal. In the future, we plan on studying those two aspects (expression and verification of probabilistic properties) while limiting ourselves to the Event-B syntax and proof mechanism.
Finally, we have not yet taken advantage of the parametric information present in (probabilistic) Event-B models. The main reason is that parameters in Event-B are used as state and/or transition variables. As a consequence, the expertise we have in the probabilistic parametric domain (as presentend in Chapters 1 and 3 for example) cannot be transfered in a trivial manner to the Event-B setting. However, two lines of future work seem promising.

— Although this would slightly restrict the expressive power of probabilistic Event-B, using distinct sets of variables for parameters used in event weights and parameters only used

in state variable assignments would allow expressing the operational semantics of probabilistic Event-B models as parametric Markov chains, therefore enabling their verification using standard techniques in this context. The next step will be to adapt those techniques to the Event-B syntax and proof mechanism.

— Parametric statistical model checking, as will be presented in Chapter 3, is a verification technique that can be applied to any parametric system that can be simulated or sampled. In particular, this could be used directly on systems expressed in the Event-B formalism, or on the translation of such systems to a programming language. Although Automated techniques to translate an Event-B specification to a C program already exist, they have not yet been extended to the probabilistic setting.

CHAPTER 3

STATISTICAL MODEL CHECKING FOR PARAMETRIC SYSTEMS

The work we present in this chapter was achieved during **Paulin Fournier**'s postdoc and **Ran Bao**'s thesis. The contributions we present here follow up on a long series of works on statistical model checking [LDB10, BBB+10a, BBB+10b, BBD+12, BBB+12, NBB+15] initiated more than 10 years ago. This line of work was initiated with the observation that, in many cases, standard probabilistic model checking techniques fail to scale up to the size of industrial systems and are inadequate when automata-based models of the systems are not easily available. As a consequence, more efficient and flexible alternative techniques have been developed. Statistical Model Checking (SMC) in particular is a simulation-based technique that offers many advantages in this context. Indeed, while its aim is not to compute the exact probabilities with which a system satisfies a given property, SMC allows to estimate this probability (or compare it with a fixed threshold) with a formally guaranteed precision and error-rate. Since it is simulation-based, SMC is not limited to automata-based models: it can be used on any executable model regardless of the formalism used to describe it. Moreover, because of its simulation-based nature, SMC is more dependant on the complexity of the property it verifies than on the complexity of the system itself. As a consequence, SMC can easily scale up to industrial-systems' size as long as they can be simulated efficiently.

However, SMC also suffers from limitations: in particular, it is mostly restricted to purely probabilistic systems, which prevents its use in the context of non-deterministic and/or parametric systems as considered in Chapters 1 and 2. In this chapter, we try to solve this problem by developing a version of *SMC for parametric systems*. In this context, instead of yielding a numeric approximation of the probability that a purely probabilistic system satisfies a property, pSMC yields a polynomial function, whose variables are the parameters of the system, that adequately approximates the probability that a parametric system satisfies the given property *for all values of the parameters*. Along with this probability, we provide parametric confidence intervals that formally guarantee a parametric precision. The technique is implemented in a prototype tool with

two versions: one that allows the parametric verification of automata-based systems described in a standard language, and a second version that allows to verify parametric Python models. While the first version is mainly used for comparison to state-of-the-art parametric probabilistic model checkers such as PRISM [KNP09], PROPhESY [DJJ+15] and Storm [DJKV17], the second one has been used on a concrete case-study where we analyze the security of a UAV flight plan while taking into account the parametric precision of its sensors and parametric wind force. For the sake of conciseness some details of the case-study and some proofs of the theoretical results have been left out of the document, but the interested reader can find them in [DFL19, BAD+19].

3.1 Introduction

Nowadays, modeling and abstracting are widely accepted as crucial steps in the understanding and study of real-life systems. In many cases, it is necessary to incorporate probabilities in the models to cope with uncertainty, to abstract complex behaviour, or to introduce randomness. Markov chains and Markov decision processes, in particular, have been widely studied as shown in Chapters 1 and 2.

Statistical model checking. Though exact verification methods are known for such models they usually require solving huge equation systems [BK08], and therefore have scalability issues with the biggest models. A way to avoid this complexity is to consider approximation techniques through simulation. In particular, Monte-Carlo simulation techniques used in the context of *statistical model cheking* [LDB10] allow to infer the real behaviour of the system via independent simulations up to a computable precision.

Parametric Markov Chains. The values given to probabilistic transitions can have a huge impact on the behaviour of the system. In the early stages of development, it may therefore be useful to have an insight on how the values of transition probabilities affect the system in order to be able to set the best value in terms of convergence speed for example. To this purpose, parametric Markov chains have been introduced in [AHV93]. They allow to replace the probability values given to transitions by parameter variables, and therefore to be able to give guarantees on the system for all possible values of the parameters.

Results. The aim of this chapter is extend the application of Monte-Carlo simulation to parametric Markov chains in order to approximate the probability of the considered property as a

polynomial function of the parameters. In addition, we also derive a confidence interval on the obtained probabilities as a polynomial function of the parameters. Aside from using parameterized models, which comes in particular with better flexibility in the modeling, robustness of the results, and usability at the earliest stages of conception, the expected benefits of our new approach are largely those of such simulation techniques for non-parameterized Markov chains: better scalability through a reduced memory footprint, a complexity that is largely independent of the model complexity (be it in terms of size of the state-space, or of used features as long as they are executable). More specific to our approach, since we derive polynomial function approximations, where exact methods lead to rational functions, these results should be easier to post-process. Finally the complexity of our approach is largely independent of the number of parameters.

In order to experimentally confirm the interest of our approach we have implemented it in a (fairly crude) prototype in Python, and we report very encouraging results on case-studies from the literature as well as a complete analysis of a case-study provided by our industrial partner PIXIEL GROUP [1].

Related work. Model checking and parameter synthesis for parametric Markov chains have been widely studied in the last decade [DJJ$^+$15, HHWZ10, BDL$^+$17, DLP16], as previously mentioned in Chapter 1. To the best of our knowledge, all existing works on this topic focus on exact techniques which either produce constraints on the parameter values [BDL$^+$17, DLP16, BDF$^+$18] (as presented in Chapter 1) or compute the probability of satisfying a given property as a rational function of the parameters [DJJ$^+$15, HHWZ10]. While these techniques have the advantage of precision, they only scale to models having few parameters. We conjecture that the approximation technique we propose in this chapter will be advantageous in the context on models with a larger number of parameters because it allows to produce polynomial instead of rational functions.

On the other hand, statistical model checking [LDB10] has, to the best of our knowledge, never been applied to parametric models as such. The closer existing techniques are reinforcement learning algorithms combined with statistical model checking, which have been applied in the context of non-deterministic and probabilistic models such as Markov decision processes [HMZ$^+$12] (where non-determinism could be replaced with parameters). While these techniques allow to compute the best (or worst) probability of satisfying a given property, they do not provide error precision or confidence intervals. Moreover, since they only compute an approximation

1. https://www.pixiel-group.com/

of the best (or worst) probability value, they do not provide a complete analysis of the effect of non-deterministic choice (or parametric transitions) on the satisfaction of the given property. This is something which is easily provided using our technique.

Section 3.2 presents our extension of statistical model checking to parametric systems. It starts with a reminder and an adaptation of some definitions from previous chapters, and an introduction of basic definitions concerning statistical model checking. Our main theoretical contribution, the adaptation of Monte-Carlo simulation to the parametric setting, is then presented in Section 3.3. In Section 3.4, we report on our prototype implementation and its use on automata-based models, and provide some comparison to other parametric model checking tools. Our main case-study is then introduced in Section 3.5, as well as its resolution using the version of our prototype tool for Python models. In Section 3.6, we give some leads for improvement of the techniques implemented in our prototype tool. Finally, Section 3.7 concludes the chapter and Section 3.8 presents perspectives for future work.

3.2 Background

We start by recalling some basic definitions, most of them already given in Chapter 1.

3.2.1 Basic definitions

As usual, the set of real numbers and the set of natural numbers are respectively written $\mathbb{R}$ and $\mathbb{N}$. Given two real numbers $a < b$, the closed, semi-open and open intervals representing all real values between a and b are respectively written $[a,b]$, $(a,b]$, $[a,b)$ and (a,b).
In this chapter, we consider Markov Chains with no labels on states (although all our results could be extended to this setting in a straightforward manner). Here, a *Markov chain* is a tuple $\mathcal{M} = (S, s_0, P)$ where S is a denumerable set of states, $s_0 \in S$ is the initial state and $P : S \times S \to [0,1]$ is the transition probability function such that for all state $s \in S$, $\sum_{s' \in S} P(s,s') = 1$.
Recall from Chapter 1 that a run of a Markov chain is a sequence of states $s_0, s_1 \ldots$ such that for all i, $P(s_i, s_{i+1}) > 0$. Given a finite run $\omega = s_0, s_1, \ldots s_n$, its length, written $|\omega|$ represents the number of transitions it goes through (including repetitions). Here $|\omega| = n$. We write $\Gamma_{\mathcal{M}}(l)$ (or simply $\Gamma(l)$ when $\mathcal{M}$ is clear from the context) for the set of all finite runs of length l, and $\Gamma_{\mathcal{M}}$ for the set of all finite runs *i.e.*, $\Gamma_{\mathcal{M}} = \cup_{l \in \mathbb{N}} \Gamma_{\mathcal{M}}(l)$. As in Chapter 1, we use the probability measure on runs based on the sigma-algebra of cylinders (see *e.g.* [BK08]). This probability

measure is written $\mathbb{P}_{\mathcal{M}}$ in this chapter. This gives us that for any finite run $\omega = s_0, s_1, \ldots s_n$, $\mathbb{P}_{\mathcal{M}}(\omega) = \Pi_{i=1}^{n} P(s_{i-1}, s_i)$. In the rest of the chapter, we only consider *finite* runs.

Example 12 (Examples of properties)**.** In this chapter, we consider properties on bounded runs and we aim at computing approximations for the following values:

Reachability $\mathbb{P}_{\mathcal{M}}(\Diamond^{\leq l} s)$**.** Recall from Chapter 1 that a run $\omega = s_0, s_1, \ldots$ is said to reach a state s in less than l steps, written $\omega \models \Diamond^{\leq l} s$, if there exists $i \leq l$ such that $s_i = s$.

Safety $\mathbb{P}_{\mathcal{M}}(\Box^{=l} E)$**.** A run $\omega = s_0, s_1, \ldots$ is said to be safe for a set of states $E \subseteq S$ during l steps, written $\omega \models \Box^{=l} E$, if for all $i \leq l$, $s_i \in E$.

Expected reward $\mathbb{E}^{l}_{\mathcal{M}}(r)$**.** Given a reward function $r : \Gamma(l) \to \mathbb{R}$ we write $\mathbb{E}^{l}_{\mathcal{M}}(r) = \sum_{\omega \in \Gamma(l)} \mathbb{P}_{\mathcal{M}}(\omega) r(\omega)$ for the expected value of r on the runs of length l.

Notice that for any property $\varphi \subseteq \Gamma(l)$, $\mathbb{P}_{\mathcal{M}}(\varphi) = \mathbb{E}^{l}_{\mathcal{M}}(\mathbb{1}_{\varphi})$ where $\mathbb{1}_{\varphi}$ is the reward function such that $\mathbb{1}_{\varphi}(\omega) = 1$ if $\omega \in \varphi$ and 0 otherwise. In the following of the chapter we will thus focus on properties of the form $\mathbb{E}^{l}_{\mathcal{M}}(r)$.

Given two Markov chains $\mathcal{M}^1 = (S^1, s_0^1, P^1)$ and $\mathcal{M}^2 = (S^2, s_0^2, P^2)$ we say that $\mathcal{M}^1$ and $\mathcal{M}^2$ have the same structure if $(S^1, s_0^1) = (S^2, s_0^2)$ and for all state $s, s' \in S^1$, $P^1(s, s') > 0$ if and only if $P^2(s, s') > 0$.

3.2.2 Parametric Markov chains

We now recall the context of Parametric Markov Chains, in a slightly modified version of the one given in Chapter 1.

Given a finite set of parameters $\mathbb{X}$ we write $Poly(\mathbb{X})$ for the set of all real (multivariate) polynomials on $\mathbb{X}$. Given a parameter valuation $v \in \mathbb{R}^{\mathbb{X}}$ and a polynomial $f \in Poly(\mathbb{X})$, the evaluation of f under valuation v is written $v(f)$.

As for Markov Chains, we omit in this chapter the labeling function of Parametric Markov Chains. In the following, a *Parametric Markov chain* is a tuple $\mathcal{M} = (S, s_0, P, \mathbb{X})$ such that S is a finite set of states, $s_0 \in S$ is the initial state, $\mathbb{X}$ is a finite set of parameters, and $P : S \times S \to Poly(\mathbb{X})$ is a parametric transition probability function.

Remark (Rational function)**.** Notice that in the definition of pMC given above, in contrast to the one given in Chapter 1, we restrict ourselves to real (multivariate) polynomials on $\mathbb{X}$. However, all our results could naturally be extended to rational functions instead, *i.e.*, functions of the form d/q where $d, q \in Poly(\mathbb{X})$.

Let $\mathcal{M}$ be a pMC and $v \in \mathbb{R}^{\mathbb{X}}$ be a valuation of the parameters of $\mathcal{M}$. Let P_v be the transition probability function obtained under valuation v, *i.e.*, $P_v(s,s') = v(P(s,s'))$ for all $s,s' \in S$. We say that v is a *valid parameter valuation* with respect to $\mathcal{M}$ if the tuple (S, s_0, P_v) is a Markov chain, *i.e.*, v defines valid probability distributions for the transitions of $\mathcal{M}$. If v is a valid parameter valuation w.r.t $\mathcal{M}$, the resulting Markov chain is written $\mathcal{M}^v$.

Remark (Consistency)**.** Notice that one can obtain the set of all valid parameter valuations as the result of a set of constraints stating that each transition has a probability between 0 and 1 and that the sum of outgoing transition probabilities is 1 for all states. The problem asking whether a pMC admits valid parameter valuations, and computing them is called the *consistency problem* (see Chapter 1). In this chapter, we do not address the consistency problem further.

Given a pMC $\mathcal{M}$, a run ω of $\mathcal{M}$ is a sequence of states $s_0, s_1, \ldots$ such that for all $i \geq 0$, $P(s_i, s_{i+1}) \neq 0$ (*i.e.*, the probability is either a strictly positive real constant or a function of the parameters). As for Markov chains we write $\Gamma_{\mathcal{M}}(l)$ for the set of all finite runs of length l and $\Gamma_{\mathcal{M}}$ for the set of all finite runs.
Observe that for any valid parameter valuation v, $\Gamma_{\mathcal{M}^v}(l) \subseteq \Gamma_{\mathcal{M}}(l)$ since v may assign 0 to some transition probabilities.

Example 13. A pMC $\mathcal{M}_1 = (S, s_0, P, \mathbb{X})$ is given as an example in Figure 3.1. In this figure, we have $S = \{0 \ldots 4\}$, $s_0 = 0$, and $\mathbb{X} = \{p, q, r\}$. As depicted, some of the transitions are parametric. For instance, $P(1,0) = p$. Let v be the parameter valuation such that $v(p) = v(q) = 0.5$ and $v(r) = 0$. According to the definition above, v is a valid parameter valuation for $\mathcal{M}_1$. Indeed, under this parameter valuation, all the transitions have values between 0 and 1 and the probabilities of the outgoing transitions of all states sum up to 1. The resulting Markov chain $\mathcal{M}_1^v$ is given in Figure 3.2.
An example of run in $\mathcal{M}_1$ is $\omega = 0,1,2,2$. The length of ω is $|\omega| = 3$. As explained above, remark that ω is not a run of $\mathcal{M}_1^v$ because the probability of the transition going from 1 to 2 has been set to 0 due to the parameter valuation v. On the other hand, all runs of $\mathcal{M}_1^v$ are also runs of $\mathcal{M}$.

3.3 Approximation in parametric Markov chains

We now move to the main result of this chapter: a simulation-based method for approximate verification of parametric Markov chains based on Monte-Carlo. We start by recalling the central limit theorem, which is at the heart of our approach.

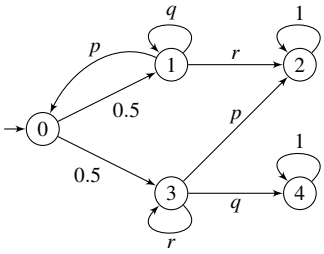


Figure 3.1 – pMC $\mathcal{M}_1$

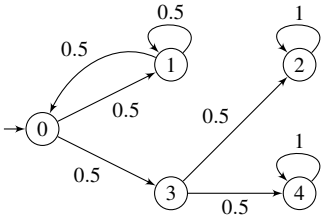


Figure 3.2 – MC $\mathcal{M}_1^v$ for parameter valuation v such that $v(p) = v(q) = 0.5$ and $v(r) = 0$

Theorem 4 (The central limit theorem (see *e.g.* [Ros09]))**.** *Let $X_1, X_2, \ldots$ be a sequence of independent and identically distributed random variables, each having mean γ and variance σ^2. Then the distribution of*

$$\frac{X_1 + \cdots + X_n - n\gamma}{\sigma\sqrt{n}}$$

tends to the standard normal distribution as $n \to \infty$. That is, for $-\infty < a < \infty$,

$$\lim_{n\to\infty} \mathbb{P}\left(\frac{X_1 + \cdots + X_n - n\gamma}{\sigma\sqrt{n}} \leq a\right) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{a} e^{-x^2/2} dx.$$

We now recall standard Monte-Carlo before presenting our contribution.

3.3.1 Standard Monte-Carlo analysis

The aim of this chapter is to propose a statistical verification method, based on Monte-Carlo, for approximating the expected value of a given reward function r on the runs $\Gamma(l)$ of a given pMC $\mathcal{M}$. In order to provide some intuition, we briefly recall how standard Monte-Carlo analysis works in the context of statistical model checking of Markov chains. In this context, a set of n samples of the runs of the MC is produced. Each of these samples is evaluated, yielding a value 1 if it satisfies the desired property and 0 otherwise. According to the central limit theorem, the mean value of the samples provides a good estimator of the probability that a random run satisfies the desired property. Moreover, the central limit theorem provides a confidence interval that only depends on the number of samples (provided this number is large enough).

Unfortunately, as the transition probabilities are not known a priori in the context of pMCs, this technique cannot be applied directly (since we cannot produce samples according to the parametric transition probabilities). The method we propose in the following is in line with a technique called *importance sampling* (see [RK16] for a description). The purpose of this

technique is to sample a stochastic system using a chosen probability distribution (which is not the original distribution present in this system) and "compensate" the results using a *likelihood ratio* in order to estimate a measure according to the original distribution. In the context of SMC, importance sampling has mainly been used in order to estimate the probability of rare events [BHP12] and/or to reduce the number of required samples in order to obtain a given level of guarantee [JLS12]. It has also been used in the context of parametric continuous-time Markov chains in order to estimate the value of a given objective function on the whole parameter space while using a reduced number of samples [BMS16]. However, to the best of our knowledge, importance sampling has never been used in order to produce symbolic functions of the parameters as we do here.

The intuition of this method is to fix the transition probabilities to an arbitrary function f, which we call *normalization function*, and to use these transition probabilities in order to produce samples of the pMC $\mathcal{M}$. However, instead of evaluating the obtained runs by directly using the desired reward function r, we define a new (parametric) reward function r' that takes into account the parametric transition probabilities. We show that, under any parameter valuation v, the evaluation of the mean value of r' on the set of samples is a good estimator for the expected value of the reward r on $\mathcal{M}^v$. The central limit theorem also allows to produce parametric confidence intervals. Contrary to standard Monte-Carlo applied to Markov chains, the precision of the obtained approximation not only depends on the size of the sample, but also on the choice of parameter valuation as well as on the chosen normalization function. These issues are further discussed in Section 3.6.

3.3.2 Parametric estimation

In the following, we introduce some notations, then move to the random variable corresponding to our new parametric reward function, and finally show the correctness of our approach. Along the rest of the section, we consider a pMC $\mathcal{M} = (S, s_0, P, \mathbb{X})$ and a reward function $r : \Gamma_{\mathcal{M}} \to \mathbb{R}$. Given a function $f : S \times S \to [0,1]$ we say that f is valid w.r.t $\mathcal{M}$ if for all states s, $\sum_{s' \in S} f(s,s') = 1$. Given a valid function w.r.t $\mathcal{M}$, let $\mathcal{M}^f$ be the MC obtained from $\mathcal{M}$ by replacing P by f. Examples of such valid functions include the evaluation of the parametric transition probability by a valid valuation, or the function $u_{\mathcal{M}}$ such that, for each state, the distribution on its successors is uniform for all successors allowed in $\mathcal{M}$ (it respects the structure of $\mathcal{M}$). In the following, f is called a *normalization function* and, in particular, $u_{\mathcal{M}}$ is called the *uniform* normalization function. The choice of a good normalization function is discussed in Section 3.6.

Let $Pa : \Gamma_{\mathcal{M}} \to Poly(\mathbb{X})$ be a parametric reward function defined inductively as follows: $Pa(s_0) =$

1 and for all run $\omega \cdot s \cdot s' \in \Gamma_{\mathcal{M}}$, $Pa(\omega \cdot s \cdot s') = Pa(\omega \cdot s)P(s,s')$. Note that Pa can be seen as the counterpart of $\mathbb{P}$ for parametric Markov chains. Indeed for any valid valuation v and any run $\omega \in \Gamma_{\mathcal{M}^v}$ we have

$$\mathbb{P}_{\mathcal{M}^v}(\omega) = v(Pa(\omega)). \tag{3.1}$$

We now define the parametric reward function r' that will allow us to estimate the expectation of r. Given any valid normalization function f and any run $\omega \in \Gamma_{\mathcal{M}}$, let r' be such that

$$r'(\omega) = \frac{Pa(\omega)}{\mathbb{P}_{\mathcal{M}^f}(\omega)} r(\omega).$$

We now prove our main result. Let $\omega \in \Gamma_{\mathcal{M}^f}(l)$ be a random sample of $\mathcal{M}^f$ and let Y be the random variable defined as follows $Y = r'(\omega) = \frac{Pa(\omega)}{\mathbb{P}_{\mathcal{M}^f}(\omega)} r(\omega)$.
The following computation shows that, under any valid parameter valuation v such that $\mathcal{M}^f$ and $\mathcal{M}^v$ have the same structure, we have $v(\mathbb{E}(Y)) = \mathbb{E}^l_{\mathcal{M}^v}(r)$.

$$\begin{aligned}
v(\mathbb{E}(Y)) &= v\left(\sum_{\omega \in \Gamma_{\mathcal{M}^f}(l)} \mathbb{P}_{\mathcal{M}^f}(\omega) Y\right) && (3.2)\\
&= v\left(\sum_{\omega \in \Gamma_{\mathcal{M}^f}(l)} \mathbb{P}_{\mathcal{M}^f}(\omega) \frac{Pa(\omega)}{\mathbb{P}_{\mathcal{M}^f}(\omega)} r(\omega)\right) && (3.3)\\
&= \sum_{\omega \in \Gamma_{\mathcal{M}^f}(l)} v(Pa(\omega)) r(\omega) && (3.4)\\
&= \sum_{\omega \in \Gamma_{\mathcal{M}^f}(l)} \mathbb{P}_{\mathcal{M}^v}(\omega) r(\omega) && (3.5)\\
&= \sum_{\omega \in \Gamma_{\mathcal{M}^v}(l)} \mathbb{P}_{\mathcal{M}^v}(\omega) r(\omega) && (3.6)\\
&= \mathbb{E}^l_{\mathcal{M}^v}(r) && (3.7)
\end{aligned}$$

(3.2) is obtained by definition of the expected value and of the distribution of Y; (3.3) is obtained by definition of the random variable Y; (3.4) is direct because we only consider runs ω such that $\mathbb{P}_{\mathcal{M}^f}(\omega) \neq 0$; (3.5) is a consequence of (3.1); and finally, since $\Gamma_{\mathcal{M}^v}(l) = \Gamma_{\mathcal{M}^f}(l)$ because $\mathcal{M}^v$ has the same structure as $\mathcal{M}^f$ we obtain (3.6).

Our adaptation of the Monte-Carlo technique for pMC is thus to estimate the expected value of

Y in order to obtain a good estimator for the expectation of r.

Let $\omega_1, \ldots, \omega_n$ be a set of n runs of length l of $\mathcal{M}^f$. Let Y_i be the random variable with values in $Poly(\mathbb{X})$ such that $Y_i = r'(\omega_i)$. Notice that the Y_i are independent copies of the random variable Y. The random variables Y_i are therefore independent and identically distributed. Let γ be the parametric function giving their mean value and σ^2 be the parametric function giving their variance.

By the results above, for all valid parameter valuation v such that $\mathcal{M}^v$ and $\mathcal{M}^f$ have the same structure, $\mathbb{E}^l_{\mathcal{M}^v}(r) = v(\mathbb{E}(Y)) = v(\mathbb{E}(\sum_{i=1}^n Y_i/n)) = v(\gamma)$. Our parametric approximation of the expected value is therefore

$$\widehat{\gamma} = \sum_{i=1}^{n} Y_i/n.$$

3.3.3 Parametric confidence intervals

We now use the central limit theorem in order to compute the *confidence intervals* associated to this estimation. As for the estimation $\widehat{\gamma}$ itself, the obtained confidence intervals will be given by parametric functions.

Since the random variables (Y_i) are independent and identically distributed, each having mean γ and variance σ^2, the expected value and variance of their sum $Y_1 + \cdots + Y_n$ is as follows: $\mathbb{E}(Y_1 + \cdots + Y_n) = n\gamma$ and $Var(Y_1 + \cdots + Y_n) = n\sigma^2$. Recall that both γ and σ are parametric functions. Let Z be the (parametric) random variable such that

$$Z = \frac{(Y_1 + \cdots + Y_n - n\gamma)}{\sqrt{n\sigma^2}}.$$

By the central limit theorem, $v(Z)$ tends toward the standard normal distribution $\mathcal{N}(0,1)$, for all valid parameter valuation v such that $\mathcal{M}^v$ and $\mathcal{M}^f$ have the same structure, as $n \to \infty$. Therefore, for all valid parameter valuation v such that $\mathcal{M}^v$ and $\mathcal{M}^f$ have the same structure, assuming that n is large enough, we obtain that

$$\mathbb{P}(-z \leq v(Z) \leq z) \approx \boldsymbol{\varphi}(z) - \boldsymbol{\varphi}(-z) = 2\boldsymbol{\varphi}(z) - 1,$$

where $\boldsymbol{\varphi}(z) = \int_{-\infty}^{z} exp(-x^2/2)dx/(\sqrt{2\pi})$ is the cumulative distribution function of the standard normal distribution $\mathcal{N}(0,1)$. As a consequence, by definition of Z, we obtain that for all valid parameter valuation v such that $\mathcal{M}^v$ and $\mathcal{M}^f$ have the same structure, and for n large enough,

$$2\boldsymbol{\varphi}(z) - 1 \approx \mathbb{P}\left(v(\gamma) - z\frac{v(\sigma)}{\sqrt{n}} \leq v(\widehat{\gamma}) \leq v(\gamma) + z\frac{v(\sigma)}{\sqrt{n}} \right).$$

The (parametric) random interval $I = (\widehat{\gamma} - z\sigma/\sqrt{n}, \widehat{\gamma} + z\sigma/\sqrt{n})$ is called a *confidence interval* for γ with level $2\boldsymbol{\varphi}(z) - 1$. Typically we use $z = 1.96$ since $2\boldsymbol{\varphi}(1.96) - 1 = 0.95$ (or $z = 2.56$ for which the level is 0.99). According to our hypothesis, this (parametric) confidence interval is only valid for valid parameter valuations v such that $\mathcal{M}^v$ and $\mathcal{M}^f$ have the same structure.

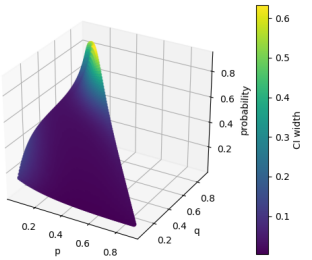
At this point, it is important to recall that the size of the confidence interval for $\widehat{\gamma}$ is also parametric: it is equal to $2 \cdot z\frac{\sigma}{\sqrt{n}}$. Indeed, the value of σ depends both on the valuation of the parameters and on the choice of the normalization function. We now explain how to compute a parametric estimation of σ. From classical probability theory, we know that an unbiased estimator for the variance σ^2 is:

$$\widehat{\sigma^2} = \frac{1}{(n-1)} \sum_{i=1}^{n} Y_i^2 - \frac{n}{(n-1)} \widehat{\gamma}^2.$$

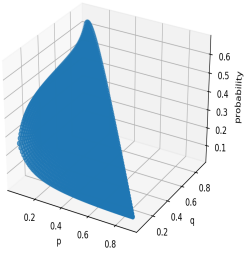
For n large enough, the (parametric) confidence interval associated to our estimation $\widehat{\gamma}$ of γ can therefore be estimated by the (parametric) interval $I = (\widehat{\gamma} - z\widehat{\sigma}/\sqrt{n}, \widehat{\gamma} + z\widehat{\sigma}/\sqrt{n})$. The (parametric) size of this interval is therefore given by $2z\widehat{\sigma}/\sqrt{n}$. Thus, for an estimation of γ with a confidence interval of level 0.95 (for a large enough value of n), the (parametric) size of the confidence interval for the estimation of the variance is $3.92\widehat{\sigma}/\sqrt{n}$. Recall that $\widehat{\sigma}$ is a parametric function that also depends on the choice of the normalization function.

3.4 Implementation

We implemented our technique in Python to validate the approach. All the following experiments have been realized on a 2,5 GHz Intel Core i7 processor. The prototype is still in the early development stages, thus we only experimented with the uniform normalization function as well as with valid valuation normalization functions. Moreover, the size of the samples is set manually. This implementation aims at validating our approach and not at competing with tools such as PARAM [HHWZ10], PROPhESY [DJJ+15] or Storm [DJKV17]. No optimizations have been implemented and we believe that this prototype could obtain much better results with simple optimizations. The code (still in development) is available at https://github.com/paulinfournier/MCpMC.



(a) With 10 000 simulations and uniform normalization



(b) Theoretical probability

Figure 3.3 – Results on pMC $\mathcal{M}_1$

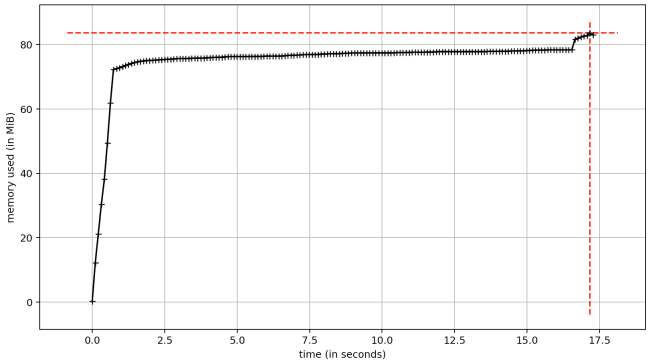


Figure 3.4 – Graphical representation of the memory consumption

3.4.1 Toy example

We first tested our program on the pMC $\mathcal{M}_1$ given in example 13. The considered property is the probability of reaching state 4. The results of our program on this example are presented in Figure 3.3a. The number of simulations was set to 10 000 and the length of the run bounded to 100. The normalization function used is the uniform normalization. We also give in Figure 3.3b the graphical representation of the theoretical probability of reaching state 4. The parsing of the model took around 3ms and the simulations took around 17 seconds. The memory consumption of the program is depicted in Figure 3.4. Note that even if the program uses around 80MB of memory, most of it is due to the loading of python libraries. A careful analysis of the memory consumption shows that the actual model only uses 0.7MB, the simulations use 6.5MB and the figure uses around 4MB. The memory consumption could thus be easily reduced by using a lighter programming language with a better handling of memory such as C.

To show that our method is transparent with regard to the number of parameters we extended this

model with 100 parameters $\{p_0, ..., p_{99}\}$. Since q remains the same and r is always $1-(p_i+q)$, this new model is equipped with a total of 101 parameters. We therefore consider an unfolding of the pMC given in Figure 3.1 of depth 101. Each time we enter a copy of state 1 or 3, new parameters are used. For example, the nth copy of states 1 and 3 use parameter $q_{n \mod 100}$ instead of q. The result of the experiment is shown in Figure 3.5. Since there are too many dimensions to plot in 2D, we only plot the result with respect to parameter q for random valuations of the parameters in (p_i). The number of simulations and simulation length are set as before. Note that the size and shade of the confidence intervals are not what one could expect but this is due to the fact that each point is evaluated for a different random valuation of the parameters in (p_i). The important point of this experiment is that the results were obtained in 21 seconds, which is approximately the same as with only 2 parameters. Note that this increase in time is due to a slight increase in the complexity of the model (the states are not really duplicated but we have to keep track of the current depth in order to consider the right parameters in each simulation).

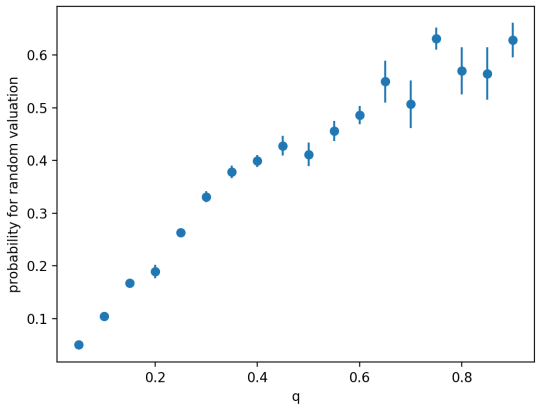


Figure 3.5 – Extension of $\mathcal{M}_1$ with 101 parameters

3.4.2 Zeroconf

The Zeroconf model, taken from the PARAM website [2], models the management of a network. When a new host joins the network, it randomly selects an address among K potential candidates. If there are already m hosts in the network the probability of collision is $q = m/K$, which is a parameter of the model. In order to detect collisions, the new host asks the others whether the address is free. If he receives a positive answer, then the new hosts considers that his address is

2. See `https://depend.cs.uni-saarland.de/tools/param/casestudies/`

valid. The second parameter of the model is the probability p that the new hosts do not receive an answer. In this case, the new host retries at most n times (here $n = 140$). In case the new host does not receive an answer after n attempts, it decides that its address is valid. We consider the expected number of attempts until the address is considered valid (either because of positive answer or because of the absence of answers after n attempts). In Figure 3.6 we present the results of our approach for 10 000 simulations, the uniform normalization function and for a simulation length of 500. Using our prototype, the experiment took around 60s. In Figure 3.6, we present as well the result obtained with PARAM (taken from their website). Remark that the shape of the distribution we obtain is similar to the one obtained with PARAM. The size of the confidence intervals are given as the color of the points.

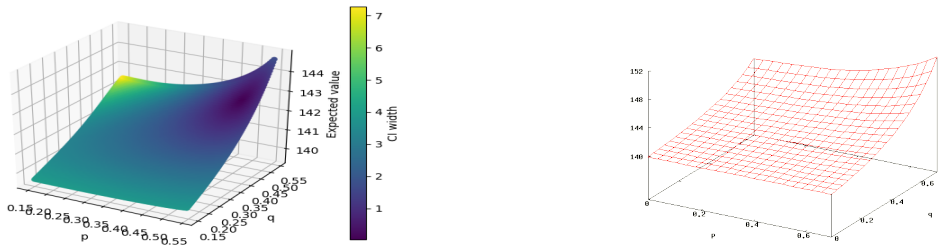


Figure 3.6 – Results for the zeroconf model obtained with our implementation (left) and taken from the PARAM website (right)

3.4.3 The crowds protocol

The Crowds protocol [RR98] aims at preserving anonymity of Internet users. To do so, each message is sent via random routes, with the assumption that a corrupted router can only see the local sender of a message. This protocol guarantees that the probability of a corrupted router observing the real sender (and not just routing another user's message) is small. A model of this protocol as been proposed in Prism [Shm04] and later extended with two parameters in [HHWZ10]. One parameter PF represents the probability of forwarding a message to a random selected member (and therefore the probability of delivering the message directly to the receiver is $1 - PF$). The other parameter $badC$ represents the probability of a router being corrupted. The reward function we are interested in corresponds to the probability of dishonest router observing the real sender more than other crowd members.

Figure 3.7 represents the output of our approach on this model for 10 000 simulations, using the uniform normalization function. This experiment was done with the same values as used on

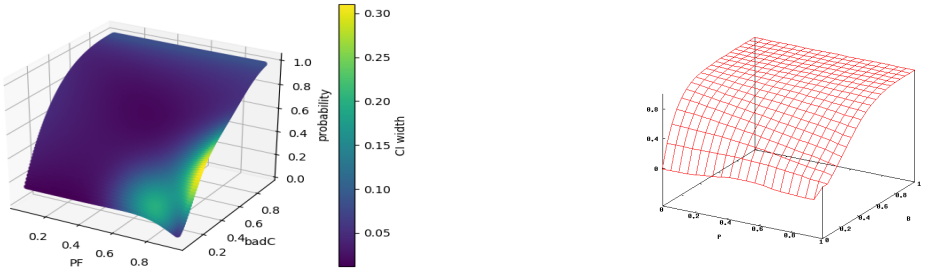


Figure 3.7 – Results on the Crowds protocol model for our implementation (left) and from the PARAM website (right)

the PARAM website, that is 5 honest crowd members and 7 different path reformulates [3]. The simulation time took around 6 minute and used around 80MB (again most of it due to loading of python libraries). Again, one can observe that the shape of the obtained distribution is similar to the one obtained with PARAM. The only zones where the two distributions are slightly different are those where the confidence intervals we produced are the largest.

3.5 Application to UAV flight plan analysis

One of the advantages of SMC is that the model under study does not need to be represented in a fully detailed manner as a MC or pMC: any symbolic representation that allows runs to be sampled is sufficient. As a consequence, the model does not have to be written in the prism language for instance, which has limitations on the use of real-valued variables and constants.
We have therefore developed a version of the prototype tool presented in the previous section that can take as an input any program written in Python that follows conditions ensuring that the underlying model is a pMC. In particular, this allows taking advantage of all the algorithmic power of python libraries in the description of a model, which is not easily done in standard model checking tools such as PRISM [KNP09], PARAM [HHWZ10], Prophesy [DJJ+15] or Storm [DJKV17].
In this section, we describe the application of the parametric statistical model checking technique presented in the previous sections to a complete case-study targeting the analysis of the trajectory of a UAV while taking into account parameters representing the precision of some of its sensors as well as some environment conditions.

3. See `https://depend.cs.uni-saarland.de/tools/param/casestudies/` for details

In the following, we start with a presentation of the context of the case-study. We then move to a description of the process we followed in order to build the model describing the UAV in this context. Finally, we present experimental results obtained using the Python version of the tool presented in the previous section.

3.5.1 Context of the case-study

Unmanned Aerial Vehicles (UAV) are more and more present in our lives through entertainment or industrial activities. They can be dangerous for their environment, for instance in case of a failure when an UAV (aka a drone) is flying above a crowd. Unfortunately until today, there does not exist any kind of UAV regulation around the world. Only some recommendations are used; for instance in order to avoid accidents in case of malfunctioning, a drone should never fly above a crowd.
Guaranteeing that a drone never flies above a crowd requires to pay close attention to the drone trajectory computation as well as to the accuracy of the measurements concerning its immediate position in space and its movements. However, a rigorous study is necessary to ensure reliability of the drone control system, for instance by decreasing the risks of failure using the appropriate tuning of the drone flying parameters which impact the computation of its trajectory. Accordingly, the questions are *how to prove that the UAV failure probability is low and which parameters have to be taken into account to ensure human safety during performances including UAVs.*
High-quality aircrafts such as Hexarotors can easily avoid the majority of minor failures related to hardware because they can fly with only five motors and the probability of concurrent failure of more than two motors is in general insignificant. In the same way, in case of battery failure, the UAV is able to land down on a specified area without any safety issue for the environment as long as it is situated in a safe zone where humans are not endangered. However, software failure may be a lot more problematic and complex to study. In this case, the UAV behavior might become unpredictable. One critical issue in this context is the potential inaccuracy of position estimation in drone systems, either as a result of inaccurate sensor measurements or of misinterpretation of data coming from those sensors. Moreover, besides aircraft system failure consideration, there is also a far more critical aspect to take into account: the weather environment. Therefore, a general approach to improve UAV safety is to study the impact of inaccuracy in position measurements on the resulting flight path compared to a given, fixed, flight plan while taking into account weather conditions.
The purpose of this case-study is therefore to provide means to study the reliability of UAVs in the context of a given flight plan. In order to do that, we have to build a formal model which

will allow us (1) to analyze the drone system and detect the most important parameters, and (2) to tune those parameters in order to reduce the system failure probability. To this intent, we thoroughly study the UAV system, formalize it and analyze it using parametric statistical model checking. Among the components of a drone system, we particularly focus on the *Flight Controller* (FC), which is responsible for computing estimations of the UAV position during its flight in order to adapt its trajectory to a given predefined flight plan. We therefore build a formal model of the flight controller in terms of parametric probabilistic models that takes into account the potential inaccuracy of the position estimation. Since UAVs are particularly sensitive to the weather environment (and in particular to wind conditions), we also enhance our model in order to take into account potential wind perturbations. Since wind force can drastically vary from one point of a given flight plan to another, we also use parameters to encode the wind force and allow our model to adapt to particular weather conditions.

3.5.2 Building a formal model of a UAV

We now present our method to build the UAV model. Recall that we are interested in studying human safety w.r.t. UAV flights. In order to do so, we use inputs such as the intended flight plan of the UAV in order to partition the spatial environment into zones that can either be safe or unsafe for the public. The aim of our model is then to evaluate the probability for a UAV to enter a "forbidden" (*i.e.*, unsafe) zone.
We start by explaining how the zones are computed with respect to the given flight plan. We then show how the UAV software can be decomposed into components and focus on the most important ones. Finally, we detail how the formal models for the important components are built and present the resulting global model.

3.5.3 Safety zones

In the context of software, considerations in airborne systems and equipment certification (named DO-178C) defined five levels of safety zones. We therefore chose to adapt this definition to our context and also partitioned the spatial environment of our drones into five zones, the most secure being Zone 1 and the most dangerous being Zone 5. These zones are characterized by their distance from the intended flight plan, as shown in Figure 3.8.

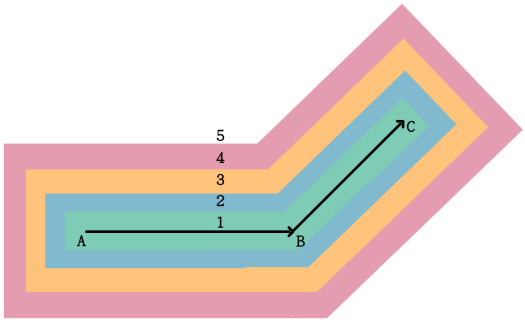


Figure 3.8 – Safety zones

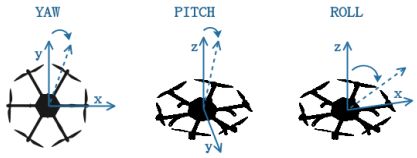


Figure 3.9 – Attitude coordinates

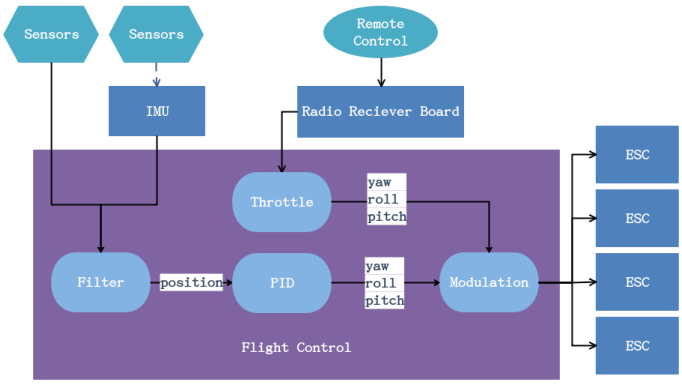


Figure 3.10 – Flight Control overview

The size of each safety zone is not definitely fixed; it can be defined for a specific requirement or for a given application. In practice the safety zones are specifically defined for a flight environment and for a given flight plan. The main principle is that no human should be present in Zones 1 to 3, while a few people can be present in Zone 4 and most people can be present in Zone 5. As a consequence, the probability that the UAV endangers humans is directly proportional to the probability that it enters Zones 4 or 5. In the following of this section, our target will therefore be to estimate this probability.

3.5.4 Drone components

We now move to the decomposition of the UAV hardware and UAV software into components and introduce the most important component in the UAV system: the flight controller (FC). The FC is responsible for collecting data from various sensors, using this data to compute the precise *position* and *attitude* of the drone and adjust the attitude in order to follow the given flight plan to the best of its ability.

Notice the difference between position and attitude: while the position of the UAV is defined by 3-dimensional coordinates x, y and z, its attitude is the collection of *yaw*, *pitch* and *roll* measurements for the UAV compared to the vertical (see Figure 3.9). The attitude allows to control the movement of the UAV: by controlling the speed of each motor, one can control which motor will be the highest, and hence control the direction the UAV will fly to.

Flight controller. As explained above, the FC is the central component in any UAV as it is responsible for collecting data from sensors and translating them to the UAV attitude. An overview of the FC of an UAV is given in Figure 3.10. Remark that the FC can be linked to

components responsible for communicating with a remote control. While these components are necessary in order to allow a pilot to take over when the automatic flight mode of the UAV fails, we will consider in the following that this is not the case and that the UAV we study are always in automatic flight mode (*i.e.*, following the pre-defined flight plan).
As one can see from Figure 3.10, the intuitive behavior of the FC is as follows. The filter uses sensors measurements in order to compute the current drone position and attitude. Since the data can be noisy and inaccurate, the filter uses complex algorithms in order to clean the noises in the measurements and compute a realistic position and attitude. Remark that in some cases, the filter can itself introduce inaccuracy in the computed position and attitude, which can be problematic. Once the estimated current position and attitude are computed, the Proportional Integral Derivative (PID) uses this information to compute the local trajectory that the drone has to follow in order to be as close as possible to its intended flight plan. This local trajectory is then transformed into a new value for the attitude of the drone. Finally, Modulation transforms this attitude into signal to the Electronic Speed Controller (ESC) which is responsible for controlling each motor's speed.

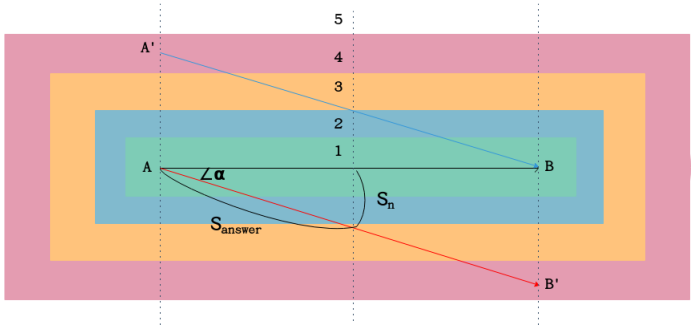


Figure 3.11 – Issue on drone location and misleading positions

Recall that we are interested in computing the probability that a UAV enters a forbidden zone while following its flight plan. By construction, as long as the position and attitude measurements are perfect, there is no reason why the UAV should deviate from its intended trajectory, and therefore the probability that it enters a forbidden zone is null. However, as explained above, the data gathered from sensors can be noisy and inaccuracy can sometimes be introduced through filtering. In this case, the estimated position and attitude of the UAV can be faulty, resulting in a deviation from the intended flight plan and potentially leading to a forbidden zone. It is therefore of paramount importance to study how the filters work and to take into account in our formal model the potential inaccuracy of position and attitude measurement. This is illustrated in Figure 3.11, where we assume that the effective position of the drone is in A, but the estimated

position that takes into account sensor and filter noise is in A'. In this case, the role of the FC is to ensure that the drone goes to B despite the current deviation from its flight plan. As a consequence, the orders sent to the ESC aim at following the trajectory from A' to B. However, since the effective position of the drone is in A, this will lead the drone to follow the trajectory from A to B' instead, therefore leading to an effective deviation from the flight plan.
For the sake of conciseness, we skip the details of the trajectory computation (they can be found in [BAD+19]), but the frequency and the accuracy of the position estimation together with details of the given flight plan (position and schedule of intermediate points) allows us to compute the effective trajectory of the drone. These computations have been taken into consideration in a global model which we now describe.

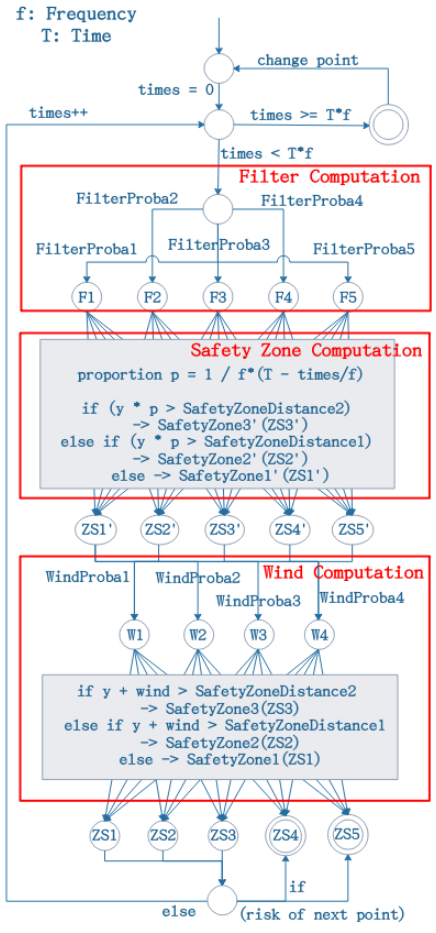


Figure 3.12 – Global behaviour of the FCS

Resulting global model. The global model of the UAV flight control system is depicted in Figure 3.12. The purpose of this model is to represent the computations taking place in the FCS in order to adapt the UAV trajectory to the intended flight plan according to inaccurate position and attitude estimations as well as wind perturbations. In this model, the exact position of the UAV is encoded using 3-d coordinates. These coordinates are then compared to the intended flight plan in order to decide to which safety zone they belong. As soon as the UAV reaches one of the forbidden zones (4 or 5), the computation stops.
The model uses several probabilistic parameters. Parameters FilterProba1, FilterProba2, FilterProba3, FilterProba4 and FilterProba5 represent the accuracy of the position and attitude estimation by both the filter and the sensors. The resulting probabilistic choice depicted in the box labelled **Filter Computation** therefore dictates the distance between the exact and estimated position of the UAV. This choice is followed by a computation in the box labelled **Safety Zone Computation** that computes the exact coordinates of the next position of the drone and allows to decide the safety zone to which this position belongs. When the wind is not taken into account, the result of this computation is enough to decide whether the model should pursue its execution. When the wind is taken into account, another step

follows, depicted in the box labelled **Wind Computation**, where other probabilistic parameters are used in order to decide the wind strength (we assume that the direction is constant) and a new position taking into account these perturbations is computed. Finally, the zone to which this last position belongs is computed and, depending on whether this zone is safe, the model goes on to another position estimation.
Remark that the position estimation frequency and the position and distance of checkpoints in the flight plan are given as inputs to the model. The position of checkpoints in the flight plan allows to compute the required UAV speed, while the frequency of the position estimation allows to fix the number of position estimations that will happen in a given flight plan (i.e. the number of loops the model goes through, at most).

3.5.5 Implementation, experimentations and results

The model presented above has been implemented in a Python program, and takes into account deviations on axes x and y, potential change of intermediate checkpoints in case of delay along a complex (more than 2 points) trajectory as well as wind perturbations. For the sake of simplicity, we assumed here that the wind direction was constant *with respect to the trajectory* (*i.e.*, wind angle is constant in the local Cartesian system associated to the drone), but the generalisation to a varying wind direction is straightforward.

Remark. Preliminary versions of our model (that do not take all details into account) have been implemented in the Prism input format and run on Prism and PARAM. In all cases but the most simplistic versions (discretized position with only 5 possible values), this has resulted in *Out of Memory* or *Timeout* errors.

As in the previous section, running our prototype tool on this model returns *polynomials* representing (1) the estimations of the probability that the drone enters an unsafe region during its flight and (2) the associated confidence interval.
Instead of reporting the resulting polynomials, which consist in thousands of terms, we will only present the evaluation of these polynomials using realistic values for the parameters. We defined two scenarios (Scenario 1, Scenario 2) with one set of values of parameters for each scenario. For these two scenarios, ProbaF0 (resp F1, F2, F3, F4) models the probability that the estimated position is from 0 to $2m$ (resp. $2-4m$, $4-6m$, $6-8m$, $8-10m$) from the real position. In the first (resp. second) scenario, we have set these values to $0.15/0.3/0.4/0.1/0.05$ (resp. $0.1/0.25/0.35/0.2/0.1$). According to field experiments, the first scenario is more realistic than the second one. Similarly, the wind parameters correspond to the probability of having a wind

Table 3.13 – Results of the experiments

	Model	**#p**	**10k**		**20k**		**50k**	
			V1	**V2**	**V1**	**V2**	**V1**	**V2**
Running time	M_1		28s		51-54s		142-143s	
Scenario 1	M_1	5	4.99%	5.09%	4.74%	5.10%	4.91%	4.98%
Conf. interv.	M_1	5	±0.85%	±0.82%	±0.55%	±0.56%	±0.36%	±0.37%
Scenario 2	M_1	5	10.38%	10.04%	9.82%	10.05%	9.95%	9.81%
Conf. interv.	M_1	5	±1.15%	±1.12%	±0.79%	±0.80%	±0.51%	±0.51%
Running time	M_2		28s		53-54s		149-155s	
Scenario 1	M_2	5	5.44%	5.31%	5.61%	5.21%	5.59%	5.47%
Conf. interv.	M_2	5	±0.98%	±0.86%	±0.69%	±0.64%	±0.42%	±0.43%
Scenario 2	M_2	5	10.8%	10.9%	10.8%	10.8%	10.9%	10.7%
Conf. interv.	M_2	5	±1.35%	±1.32%	±0.91%	±0.92%	±0.57%	±0.57%
Running time	M_3		185-190s		311-314s		612-621s	
Scenario 1	M_3	9	4.95%	5.97%	5.28%	6.62%	4.16%	5.61%
Conf. interv.	M_3	9	±5.22%	±5.71%	±4.71%	±6.25%	±1.86%	±4.38%
Scenario 2	M_3	9	9.55%	9.87%	10.3%	11.3%	9.57%	10.7%
Conf. interv.	M_3	9	±8.40%	±7.86%	±7.04%	±7.89%	±5.29%	±3.99%

force of $0-20km/h$,$20-30km/h$, $30-50km/h$ and $50-70km/h$ respectively and have been set to $0.55/0.43/0.01/0.01$ (which corresponds to typical weather conditions in Nantes, France) for the numerical evaluation. In both scenarios, Zone 4 (resp. 5) is situated $8m$ (resp. $50m$) from the flight plan.

Remark. Although we only provide the evaluation of the polynomials for two scenarios, the advantage of our technique is that we could evaluate the polynomials for any given parameter valuation. Indeed, once the polynomials are obtained, their evaluation with a given parameter value is very efficient: no further simulations of the model are given. On the contrary, using standard SMC techniques would require performing again the whole set of simulations every time a new parameter valuation is chosen.

In Table 3.13, we gather the results for running the simulation for the two considered scenarios; the simulation with pSMC is performed with 10k, 20k and 50k samples. Each time, the two polynomials are computed and then evaluated using the parameter values given above. In order to illustrate the stability of our results despite their statistical nature, each complete scenario was performed two times (labelled V1 and V2 in the table). The value reported in the table represents

the probability of the UAV eventually reaching Zones 4 or 5 during its flight. Experiments were performed using three different versions of the model. M_1 is a simple version that does not take wind perturbations into account. M_2 is an intermediate version where wind force is probabilistic but not parametric (the values are set to the scenario given above). Finally, M_3 is the complete version where wind force is parametric. For each model, the number of parameters involved is reported in the column $\#p$. The flight plan consists of only 3 points, resembling the one shown in Figure 3.8, with a total flight duration of $5s$ and a position estimation frequency of $1Hz$. Remark that the probabilities of entering the forbidden zones are quite high. This is not surprising as Zone 4 is situated $8m$ from the intended trajectory and the precision of position estimation can be up to $10m$. These values have been made deliberately high for the purpose of this study but can be chosen more realistically when verifying the real model.

Although the analysis of the obtained polynomial is not an easy task, the user can use those polynomials to get an estimation of the probability that the drone enters the forbidden zones for any chosen parameter values *without having to run the model again*. More than that, this case-study has confirmed the potential of parametric statistical model checking and shown that our prototype tool, even in its current crude form, can scale up to systems well out of reach of state-of-the-art (parametric) model checkers.

However, there are still many open questions on how to analyze and take advantage of the results offered above (besides the evaluation of the polynomials). Moreover, many choices in our technique have been made by default, but could have been made otherwise. In the next section, we discuss some promising lines of work to enhance the parametric statistical model checking technique itself.

3.6 Potential improvements

In this section we explore some variants of our technique in order to tackle some of its inherent problems. As a reminder of the notations and definitions given in Section 3.3, the random variable representing the binary output of each of simulations is written Y and the normalization function f is the one that assigns values to transition probabilities in order to sample the considered pMC $\mathcal{M}$.

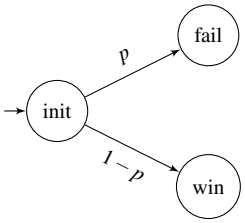


Figure 3.14 – A simple pMC.

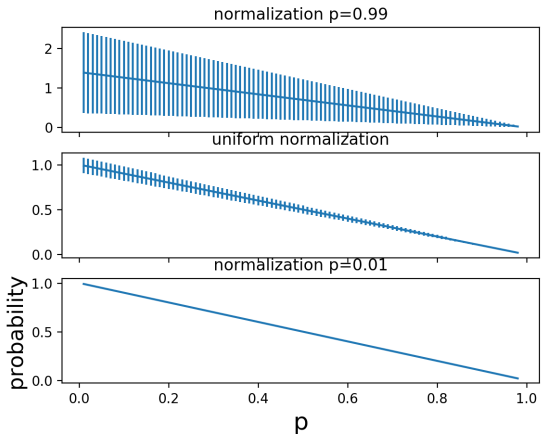


Figure 3.15 – Impact of the choice of the normalization function f on the size of confidence intervals

3.6.1 Choice of normalization function

The choice of the normalization function f has a huge impact on the convergence speed of our method. Figure 3.15 illustrates this on a simple example where the property tested is the reachability of state *win* in the three state pMC given Figure 3.14. In Figure 3.15, where the number of simulations is set to 500, we can see that with a normalization function assigning 0.99 to p the confidence intervals are much larger (in particular for small values of p) than with a normalization function assigning 0.01 to p.

This can be explained by the following computation that bounds the variance of Y.

$$v(Var(Y)) = v\left(\mathbb{E}^l_{\mathcal{M}^f}(Y^2) - \mathbb{E}^l_{\mathcal{M}^f}(Y)^2\right) \tag{3.8}$$

$$= v\left(\sum_{\omega\in\Gamma_{\mathcal{M}^f}(l)} \mathbb{P}_{\mathcal{M}^f}(\omega)y(\omega)^2\right) - \mathbb{E}^l_{\mathcal{M}^v}(r)^2 \tag{3.9}$$

$$= \sum_{\omega\in\Gamma_{\mathcal{M}^f}(l)} (v(Pa(\omega)))^2 r(\omega)^2/\mathbb{P}_{\mathcal{M}^f}(\omega) - \mathbb{E}^l_{\mathcal{M}^v}(r)^2 \tag{3.10}$$

$$= \sum_{\omega\in\Gamma_{\mathcal{M}^f}(l)} \mathbb{P}_{\mathcal{M}^v}(\omega)r(\omega)^2\mathbb{P}_{\mathcal{M}^v}(\omega)/\mathbb{P}_{\mathcal{M}^f}(\omega) - \mathbb{E}^l_{\mathcal{M}^v}(r)^2 \tag{3.11}$$

$$\leq m \sum_{\omega\in\Gamma_{\mathcal{M}^f}(l)} \mathbb{P}_{\mathcal{M}^v}(\omega)r(\omega)^2 - \mathbb{E}^l_{\mathcal{M}^v}(r)^2 \tag{3.12}$$

$$= m\mathbb{E}^l_{\mathcal{M}^v}(r^2) - \mathbb{E}^l_{\mathcal{M}^v}(r)^2 \tag{3.13}$$

$$= (m-1)\mathbb{E}^{l}_{\mathcal{M}^v}(r^2) + Var^{l}_{\mathcal{M}^v}(r) \tag{3.14}$$

Where $m = \max_{\omega \in \Gamma_{\mathcal{M}^f}(l), r(\omega)\neq 0} \mathbb{P}_{\mathcal{M}^v}(\omega)/\mathbb{P}_{\mathcal{M}^f}(\omega)$.
In the example, with the normalization function assigning 0.99 to p we have $m = (1-p)/0.01 = 100 - 100p$, thus for small values of p the obtained variance is great. On the contrary, considering the normalization function assigning 0.01 to p, we have $m = (1-p)/0.99$, which is always quite small.
Following this remark, a good direction to improve the implemented prototype would be to automatically choose the normalization function minimizing m. Note that such a normalization function can be hard to compute, but one could first use the uniform normalization function then find which valid valuation maximizes the size of the confidence interval and restart an evaluation for this valid valuation. This technique may produce good results since the size of the confidence intervals decreases only with the square root of the number of runs. Thus decreasing m may be faster than increasing the number of simulations.

3.6.2 Modification of the structure

Notice that our technique is valid only for valuations such that $\mathcal{M}^f$ and $\mathcal{M}^v$ have the same structure. Indeed, to obtain the approximation of the parametric expected value we divide by the number of runs. However, when evaluating this parametric expected value on valuations modifying the structure of the MC $\mathcal{M}^f$, it may be the case that some of the runs are impossible for this valuation. For example, any run that takes a transition with a parametric probability p is impossible to obtain for a valuation v such that $v(p) = 0$, although it may appear in the simulations of $\mathcal{M}^f$ if $f(p) \neq 0$.
To address this problem, instead of dividing by the total number of runs obtained when simulating under the normalization function f, we can divide by the number of runs that are possible for the considered valuation v. Formally, instead of the estimator $(\sum_{i=1}^{n} Y_i)/n$, we can use $(\sum_{i=1}^{n} Y_i)/N$ where N is the function of v defined by:

$$N(v) = |\{i | \mathbb{P}_{\mathcal{M}^v}(\omega_i) > 0\}|$$

This can be computed on-the-fly. Defining the estimated variance and the random confidence interval with the same technique would give us a parametric approximation technique which is valid for any valid valuation (regardless of structure preservation). Note however that this is valid only if the structure of $\mathcal{M}^f$ allows more runs than the structure of $\mathcal{M}^v$ *i.e.*, if $\Gamma_{\mathcal{M}^v}(l) \subseteq \Gamma_{\mathcal{M}^f}(l)$.

Otherwise, this technique only gives an approximation of $\mathbb{E}^{l}_{\mathcal{M}^v}(r(\omega)|\omega \in \Gamma_{\mathcal{M}^f}(l))$.
Notice also that this is useful only if $N(v)$ is large enough. Indeed if $N(v) << n$ it may be more relevant to estimate Y for a normalization function giving the same structure as $\mathcal{M}^v$.

3.6.3 Complement of the property

In classic Monte-Carlo, there is no need to consider the complement of the property. Indeed, when one approaches both the probability of φ and that of $\neg\varphi$ by $\hat{\gamma}$ and $\widehat{\gamma_\neg}$ respectively, it is always the case that $\hat{\gamma} = 1 - \widehat{\gamma_\neg}$. In our approach however, since the probabilities of the runs are normalized, this does not always hold. Consider for example the pMC given figure 3.14 with a normalization function such that $p = 1 - \varepsilon$ with ε close to 0. With high probability, our approach would give that the probability of reaching *win* is $\hat{\gamma} = 0$ since it would miss the run reaching *win* which has a really low probability for this normalization function (note that this problem only appears if the number of simulations is not big enough). However, if we estimate the negation as well, we have $\widehat{\gamma_\neg} = (n * p/(1-\varepsilon))/n = p/(1-\varepsilon)$, which is much closer to the truth for any valuation with $p \not\approx 1$.
It is thus relevant to consider both the property and its negation when considering a parametric approximation of a probability. Note that this holds only for the approximation of a probability and not for the general expected value of a reward. Note also that this is a good example of the importance of a good normalization function.
To implement this in practice, one could approach both the probabilities of φ and $\neg\varphi$ and whenever $v(\widehat{\gamma_\neg}) \not\approx 1 - v(\hat{\gamma})$ one could either increase the number of simulations or restart the approximation for the normalization function v.

3.7 Conclusion

In this chapter, we have presented a new technique for statistical model checking of parametric Markov chains. This technique is based on a parametric adaptation of the standard Monte-Carlo analysis. We have shown that our technique allows to estimate the expected value associated to any reward function by a polynomial function of the parameters, and proposed as well a parametric confidence interval for the estimation. Compared to exact model checking techniques, the technique we have proposed here offers the same benefits as standard simulation techniques for non-parametric models: better scalability and a complexity which is largely independent of the model complexity (be it in the size of the state space, in the type of used features, or

in the number of parameters). Contrary to existing statistical model checking techniques for non-deterministic systems (such as the one presented in [HMZ+12]), the one we have proposed in this chapter allows to compute parametric confidence intervals. Finally, our technique has been implemented in a prototype tool using the Python language. This prototype tool accepts inputs both in the PRISM language and in the form of Python programs. It has been tested on a set of benchmarks and on an industrial case-study with encouraging results.

3.8 Perspectives

As we have shown in this chapter, parametric statistical model checking is a technique that offers a lot of promises, but is, in its current state, far from being used at its full potential. Besides the potential improvements presented in Section 3.6, a lot of work remains in order to compare this technique (and our prototype tool) to state-of-the-art tools such as PARAM [HHWZ10], Prophesy [DJJ+15] or Storm [DJKV17]. In particular, as soon as our tool is in a more optimized state, we plan on running a set of benchmark to compare its scalability in terms of model size with the one of Storm [DJKV17]. We conjecture that such an optimized tool as Storm will be orders of magnitude more efficient than our tool on small models. However, we expect that the resources needed by Storm will grow linearly with the model size, while those needed by our tool should remain fairly constant. On the other hand, the resources needed by our tool should grow linearly with the size of the property that we verify, whereas those needed by Storm should not. Still, when considering the results presented in Section 3.5, we are confident that regardless of the property, our tool should be able to handle models that Storm cannot.

The parametric statistical model checking technique we have presented can also be adapted to verify models with non-determinism, such as Markov Decision Processes. Indeed, the non-deterministic choices in this context could be replaced by parametric choices and our technique could then be used in order to study optimal schedulers. Moreover, we could also envision using the version of our tool that takes as input a Python program to study the optimization of controllers: by considering a non-deterministic/probabilistic/parametric program, one could build a parametric controller that could then be analyzed and optimized in an automated manner by using our technique (*i.e.*, executing the controller in parallel with the system in order to be able to analyze its effect on the system). This could be done for instance on programs generated using the probabilistic Event-B formalism presented in Chapter 2.

Finally, the most exciting perspective, to our point of view, is the ability of this technique and the associated tools to analyze models and systems that are described in any programming

language. In the past years, we have indeed been in contact with researchers from other domains, such as Oceanography and we have learnt from those interactions that, although the verification techniques and the modeling formalisms we develop in our domain are very powerful and could be very useful to them, the effort that needs to be done in order to master both the modeling formalisms and the techniques is too important for potential users out of our domain. However, because of its simplicity and because it can be used on existing models written in any language without much effort, we have been able to convince some of them that statistical model checking can be integrated in the parameterization process of some of their models. Notably, we have developed a technique, called the Statistical Model Checking Engine (SMCE), that can be used almost automatically on any probabilistic and parametric model in order to *tune* the model to make it fit with experimental data. Compared to other existing parameterization techniques, the SMCE not only yields the set of *optimal* parameter values, but also provides the user with a detailed correlation analysis of the whole set of parameters – which is outside the scope of any other parameterization technique used in this context, to the best of our knowledge. This work has been published in a famous broad-audience scientific journal [REG$^+$20], which is, to our opinion, a major achievement.

Our aim now is therefore to pursue the development of both sides of our tool, in order to propose state-of-the-art automated parametric analyses that could be both used by researchers from our domain but also by any modeler able to write an executable program representing their system. Finally, we also aim in the long term at linking the analysis we do here with automated abstraction techniques based on (deep-)learning in order to reduce the cost of sampling very complex models. Succeeding in this challenge would then allow us to use our techniques to verify large-scale models such as those developed for the analysis of climate change.

CHAPTER 4

LEARNING AND VERIFICATION OF GRAPHICAL EVENT MODELS

The work we report in this chapter was achieved during **Dimitri Antakly**'s thesis, in collaboration with GFI informatique group [1]. GFI is an international group that mainly proposes computer science solutions and expertise, touching a wide variety of domains (banking, road safety, supervision, cloud, AI solutions and many more). The main expectations of GFI about this thesis were to explore behavior analytics including how it can be applied in security or supervision, and to use innovative techniques in order to be distinguished from the available market solutions. In this chapter, we tackle uncertain systems in a different form and context than those previously considered. Instead of analyzing a given model of the system of interest, we here assume that the only information we have on the system is a precomputed set of data and a security property. We therefore focus on a formalism dedicated to *learning*, along with its learning algorithm, and develop techniques for verifying the learned model and analyzing its security w.r.t. the given property.

4.1 Introduction

In most of nowadays real life jobs or applications, employees, workers, consumers and users are using connected devices. Hence, they are prone to not only external exploits but also internal misuse that can become dangerous. Consequently, it is important to monitor the behavior of users and workers in their environment to ensure a "secure" work flow. For instance, monitoring the behavior of truck drivers can allow the supervisor to verify if they are getting enough rest time while on route. Computer scientists have been putting a lot of effort in researches in order to build a safe perimeter in which connected devices can be used [WHA$^+$16, A$^+$15]. Contrarily to previous chapters, we consider here that a precise model of the systems under study is not

1. http://www.gfi.world

available. More than that, we also consider that the systems can only be analyzed through a given set of observations that cannot be extended. The uncertainty here therefore comes from the lack of information we have on the behavior of the system/user.

In this chapter, we use behavior analytics in order to classify behaviors, from "standard" to "dangerous", by rating their level of dangerousness. The key to establishing an adequate behavior analytic is a set of faithful data recorded from the concerned system. In 2020 we generate in ten minutes more data than the entire data that is recorded throughout history until 2003. Thus, clearly we have no real problems with generating data nor storing it, hence the problem is our capability of extracting useful information from this data. This is where data mining techniques [VDA14] come into play.

In order to build a secure access to data in a real world system and to ensure its safeness from any upcoming potential threat, one should learn the dependencies and behaviors of the different components of the system. One must then identify malicious behaviors and act at the right moment to intercept them. This is where model-based Machine Learning, that allows to build a model of the concerned system/user based on observed data, comes into play. Many types of modeling formalisms exist in the literature, each developed for its own purpose. The ones that were considered in the previous chapters, such as Event-B or (parametric) (Interval) Markov Chains, are better tailored to the verification of systems whose models are available. Other families of modeling formalisms, such as Hidden Markov models [RVVDA08, KA98, VM98] or (Dynamic) Bayesian Networks [MR02], are better tailored to the learning of uncertain discrete probabilistic models. In this work, we insist on the importance of timing information in the dependencies between variables of the system, and therefore focus on continuous-time graphical probabilistic models. Several such famillies of models exist, *e.g.*, continuous time Bayesian networks [NSK02], Markov jump processes [RT13], Poisson networks [RGH05] and Graphical Event Models (GEMs) [GMX11]. In this work, we chose to use *Recursive Timescale Graphical Models* (RTGEMs) [GM16] a sub-family of GEMs, that present advantages compared to the other formalisms. In particular, they are designed to universally approximate any smooth, non-explosive, stationary, multivariate temporal marked point process [DVJ07].

In this chapter, we therefore propose a strategy for analyzing the level of dangerousness of a given system only observed through pre-generated data by using RTGEM models. The strategy unfolds as follows:

(1) We start by learning an RTGEM model that is representative of the given data;

(2) We analyze this model and decide whether it satisfies given security properties;

(3) When it does not, we explore the neighborhood of this model to find an RTGEM that does satisfy the property and, if one is found, measure the distance between this safe model and the original one. This distance gives us the level of dangerousness of the original data.

The chapter is dedicated to developping and experimenting the necessary techniques to achieve the above strategy. In Section 4.2, we start with properly introducing the formalism and methods used later on. In particular, we recall statistical model checking (already introduced in Chapter 3), which will be used in the verification process of RTGEM, and provide a progressive definition of the RTGEM formalism. Section 4.3 introduces the main techniques required for our strategy, such as the learning and sampling algorithms for RTGEM, and defines the notion of distance used for determining the level of dangerousness of a given RTGEM. Finally, Section 4.4 details the experiments performed to evaluate our strategy and Sections 4.5 and 4.6 conclude the chapter.

4.2 Background

We start with some background definitions that will then be used in the rest of the chapter. As seen in the previous chapters, model checking suffers from the state explosion problem when the models are large. In the context of this Chapter, our aim is to build models that represent complex statistical dependencies between several variables. As a consequence, the models we will build might be large. Moreover, standard models used for representing statistical dependencies (*e.g.*, Bayesian models [CHM97, MR02] or probabilistic graphical models [TEF$^+$05, GM16]) are not adapted to standard model checking techniques. As a consequence, we have chosen to build our strategy on statistical model checking [LDB10], which we now recall.

4.2.1 Statistical model checking

As intoduced in Chapter 3, SMC is a simulation-based technique that allows estimating the probability that a model satisfies a given linear property while giving formal guarantees on the precision and the error-rate of this estimation. Moreover, one of the advantages of SMC is that it can be applied to any type of stochastic model that can be executed.
SMC comes in two flavors: (1) quantitative SMC that allows to verify quantitative properties (*i.e.*, compute the probability with which the model satisfies a property); and (2) qualitative SMC that allows to verify qualitative property (*i.e.*, check whether the probability with which a

model satisfies a property is greater than a given threshold). Quantitative SMC, and in particular Monte-Carlo estimation, has been presented in details in Section 3.3.1. We now briefly present the qualitative version of SMC.

In the following, we consider a stochastic system $\mathcal{S}$ and a linear property φ. As previously mentioned, SMC is based on simulations, and each simulation is represented by a trace that has a binary outcome for satisfying the property. Therefore, we consider a discrete random variable $\mathcal{B}_i$ following a Bernoulli distribution of parameter p, that is associated to each trace and can take two values: 0 (if the property is not satisfied) or 1 (if the property is satisfied). Assume that $Pr[\mathcal{B}_i = 1] = p$ is the probability of satisfying the property and $Pr[\mathcal{B}_i = 0] = 1 - p$ is the probability of rejecting the property. Qualitative SMC allows to decide if p is greater (or smaller) than a given threshold θ.

Qualitative approach. The original qualitative approaches are proposed in [You05, SVA04], and are based on hypothesis testing. The idea is to test two hypothesis, $H_0 : p \leq \theta$ against $H_1 : p > \theta$, while bounding the probability of making an error. A Type-I error is when we accept H_0 while H_1 holds and a Type-II error is when we accept H_1 while H_0 holds. Therefore, we define two parameters α and β, with (α, β) the *"strength"* of the test, being the pair of bounding errors. An ideal performance of the test is when the Type-I error is equal to α and the Type-II error is equal to β, and both α and β are small. However, in practice it is impossible to ensure a low α and β simultaneously. In order to avoid this problem, an indifference region $[p_0, p_1]$ is defined. Let $p = \mathbb{P}_{\mathcal{S}}(\varphi)$ the probability of φ being satisfied on the system $\mathcal{S}$. In order to determine whether $p > \theta$ (qualitative property, θ being in $[p_0, p_1]$), one can test $H_0' : p \leq p_0$ (instead of $H_0 : p \leq \theta$) against $H_1' : p > p_1$ (instead of $H_1 : p > \theta$. However, the tuning of the indifference region in practice is constrained by the loss of precision (when the size of the region increases) and by the fact that we cannot conclude about the result if p is inside $[p_0, p_1]$. We present two hypothesis testing algorithms: *Single Sampling Plan (SSP)* and *Sequential Probability Ratio Test (SPRT)*.

The *Single Sampling Plan* consists in specifying a constant c and a number of simulations n to test whether $\sum_{i=1}^{n} b_i > c$, where b_i is the outcome of the Bernoulli random variable, in order to see which hypothesis is accepted. Thus, the hardest part in this algorithm is to compute values for the pair (n, c) with respect to the pair (α, β) and the indifference region. The number n increases when we minimize the size of the indifference region and the parameters α and β. An optimization algorithm was proposed in [You05] in order to determine a pair (n, c) where n is minimal.

The *Sequential Probability Ratio Test* is an approach proposed by Wald [Wal45], consisting in choosing two values A and B ($A > B$) ensuring the strength of the test and that are computed using α, β and the indifference region. Unlike SSP, this approach does not fix a number of simulations *a priori* but instead dynamically checks whether the number of simulations done so far allows to conclude. As soon as a conclusion can be obtained, the algorithm stops. If m is the number of already performed observations, the test is based on the following metric :

$$\frac{p_{1m}}{p_{0m}} = \prod_{i=1}^{m} \frac{\mathbb{P}(B_i = b_i \mid p = p_1)}{\mathbb{P}(B_i = b_i \mid p = p_0)} = \frac{p_1^{d_m}(1-p_1)^{m-d_m}}{p_0^{d_m}(1-p_0)^{m-d_m}},$$

where $d_m = \sum_{i=1}^{m} b_i$. It is shown in [You05] that we can accept H_1 after m samples if $\frac{p_{1m}}{p_{0m}} \geq A$, whereas we can accept H_0 if $\frac{p_{1m}}{p_{0m}} \leq B$. Using this approach we can reduce the number of simulations in certain scenarios compared to the Single Sampling Plan approach (where the number of samples is fixed to n), and avoid doing $n-m$ useless samples. However, it is also demonstrated that convergence can be very slow when p is too close to θ. In this case, SSP might be more advantageous.
We now progressively introduce the types of models that will be used in the rest of this chapter: *Recursive Timescale Graphical Event Models (RTGEM)*.

4.2.2 Graphical event models

We start with a formal definition of the structure we use to represent our input data.

Marked point processes

Let S be a totally ordered set. A *point process* is a set of points that are positioned (following the order) in S. A *marked point process* is a point process that contains additional features at each point. In other words, a marked point process (m.p.p.) is composed of a point process and marks associated witch each point. If we consider a set of marks M, an m.p.p. can be expressed as the pair:

$$\{(s_i, m_i) : i = 1, ..., n\}$$

where $s_i \in S$, $m_i \in M$ and $s_i \leq s_{i+1}$ for all $1 \leq i \leq n$.
Henceforth, we assume that the space S that is used is the one-dimensional space of time, and the marks are labels describing each point. Marked point processes are used for expressing

event streams in this framework, i.e. random labeled (marked) events (points) that arrive at different times $\{t_i\}$ (unidimensional space). The word point is used for describing events as being instantaneous in the time dimension and for the sake of simplicity, the notation l for labels is going to be used instead of m for marks.

In practice, we tend to use a timeline to represent data that arrive at irregular intervals. A timeline can be defined as a sequence of pairs $\{time, event\}$ capturing the relative frequency and ordering of events [WP13]. In figure 4.1, an example is shown of how a timeline can be seen as a multivariate marked point process.

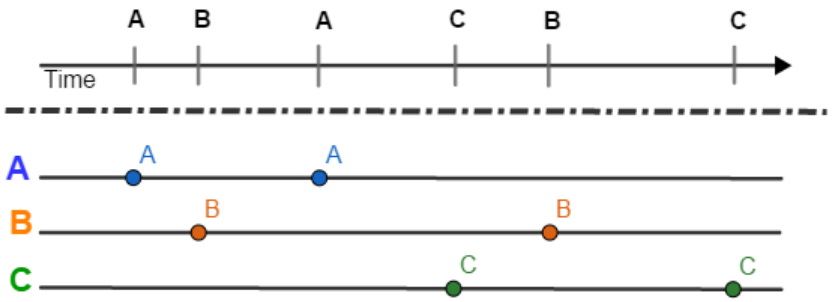


Figure 4.1 – Example showing the decomposition of a timeline into marked point processes

In the following, we define event streams as well as stochastic models that are capable of representing such an evolutionary process.

If we take a step back, a point process can be described by unrolling a stochastic model that defines inter arrival times between different events, i.e. defining the time of the next event based on all the times of previous events. In the following, we will explore a family of stochastic models based on conditional intensity functions, called *Conditional Intensity Models (CIM)*. We start by defining the data type that is generally used for learning these models.

Event streams. An *event stream* consists in a timed sequence of events with strictly increasing timestamps. An event stream can be written as: $(t_1, l_1), \ldots, (t_n, l_n)$, with $0 < t_i < t_{i+1} < t^*$ ($t^* = t_{n+1}$) for all $1 \leq i \leq n-1$ and where l_i are labels chosen from a finite set of labels $\mathcal{L}$. Hence, a single event is defined by the pair (t_i, l_i) but for the sake of simplicity we sometimes refer to an event only by its label l when the context is clear enough. We note that in the following, $t_0 = 0$ and $t^* = t_{n+1}$ are used as conventions, as well as the fact that two events cannot occur at the

exact same time. An event stream can be considered as an m.p.p. and the data that can be derived from it can be written as x_{t^*}. We write $|x_{t^*}|$ for the size of our data x_{t^*} (the number of events in the sequence, here it is n). The history at time t is the set of all the events that occurred before t: h_i denotes the ith history $h_i = (t_1, l_1), ..., (t_{i-1}, l_{i-1})$. With this in mind, we define a *conditional intensity function*. In the following, and for the sake of simplicity, data will sometimes be written as x, history as h and time as t, without indexing them explicitly at every instance, but providing enough context to understand their use.

Definition 14. A conditional intensity function, defines the risk of observing a given event at a certain time, depending on the observed history in some data x. We recall that $\mathbb{E}[X]$ is the expected value of a random variable X. Mathematically a conditional intensity function λ is written as:

$$\lambda(t) = \lim_{\Delta_t \to 0} \frac{\mathbb{E}[N(t, t+\Delta_t)|h_t]}{\Delta_t}$$

where $N(T)$ denotes the number of points (events) occurring in a time interval T in x.

In other words, the conditional intensity function allows to specify the mean inter arrival times between two events in a process. Furthermore, we recall an important assumption for the following about this approach: there cannot be more than one event in an infinitesimal interval of time.

Before moving forward to the definition of a CIM, we start with an introductory example about a common probability distribution based on an intensity function. Consider for instance the Poisson distribution [Hai73], that is a probability distribution representing a number of events occurring in a fixed interval of time under the assumption that any event is independent with regards to the duration from the previous events. A Poisson distribution is used for modeling processes such as the number of cars arriving to a garage between two given times. The probability function of a random variable X that follows a Poisson distribution of parameter λ (where also the expectation and the variance are equal to λ) is given by: $\mathbb{P}(X = k) = \frac{\lambda^k e^{-\lambda}}{k!}$. We note that in order to sample from such distribution, the inter arrival times between every two events should be computed. The inter arrival time follows the exponential distribution in the case of a Poisson distribution, with a mean of $1/\lambda$, i.e. the mean time between every two events in a Poisson distribution is $1/\lambda$.

A CIM that models a marked point process must treat each event individually, thus each label

$l \in \mathcal{L}$ must be associated with its own conditional intensity function $\lambda_l(t \mid h)$. The conditional intensity function, describes the risk of having the event l at time t given a certain history h of events (the totality of the events or a part of them depending on the CIM). For instance, in a *Markovian* CIM where an event can be particularly dependent from a set of events (called parents), the conditional intensity functions satisfy the following property:

$$\lambda_l(t \mid h) = \lambda_l(t \mid [h]_{Pa(l)})$$

where $Pa(l)$ is the set of parents of l and $[h]_{Pa(l)}$ is the history of the parents only.

Definition 15. [DVJ07] Formally, a CIM θ is a set of indexed conditional intensity functions $\{\lambda_l(t \mid x)\}_{l \in \mathcal{L}}$. The data likelihood function is the joint density function of all the points in the m.p.p. that can be factorized into a product of all conditional intensity functions and can be written as:

$$p(x \mid \theta) = \prod_{l \in \mathcal{L}} \prod_{i=1}^{n} \lambda_l(t_i \mid h_i; \theta)^{1_l(l_i)} e^{-\Lambda_l(t_i \mid h_i; \theta)}$$

where $\Lambda_l(t \mid h; \theta) = \int_{-\infty}^{t} \lambda_l(\tau \mid x; \theta) d\tau$ for the data x and the indicator function $1_l(l')$ is one if $l' = l$ and zero otherwise.

We now introduce *Piecewise-Constant Conditional Intensity Models (PCIMs)*.

Piecewise-constant conditional intensity models

Piecewise-Constant Conditional Intensity Models (PCIMs) are based on the assumption that intensity functions are constant in given time intervals. In other words, there is a mapping associated with these models that assigns a parameter λ for every label in $\mathcal{L}$ according to an *active* state. The active state is determined in function of the time t and the data x.
These models are important because they form the basis of graphical event models and they ensure fast and efficient learning and inference. These models are related to Poisson networks [TEF+05], since Poisson networks also contain piecewise constant parameters in their mapping (see more detail in [GMX11]). However, it was experimentally shown in [GMX11] that they are, by orders of magnitude, faster to train than Poisson networks.
Let $\mathcal{L}$ be a set of labels. Each label $l \in \mathcal{L}$ is associated to a set of discrete states Σ_l. For each label l and each state $s \in \Sigma_l$ we have a parameter λ_{ls}. An active state s for each label (correspondingly

the active parameter λ_{ls}) has to be determined by a mapping $\sigma_l : T \times X \longrightarrow \Sigma_l$ (with T the set of all possible times and X the set of all possible data).

Definition 16. A PCIM is defined by local structures $S_l = (\Sigma_l, \sigma_l(t,x))$ and local piecewise constant parameters λ_{ls}. Let $S = \{S_l\}_{l \in \mathcal{L}}$ be the set of all discrete states and $\theta = \{\lambda_{ls}\}_{l \in \mathcal{L}, s \in \Sigma_l}$ be the set of associated parameters. The PCIM data likelihood knowing the structure and the parameters in this case can be written as:

$$p(x \mid S, \theta) = \prod_{l \in \mathcal{L}} \prod_{s \in \Sigma_l} \lambda_{ls}^{M_{ls}(x)} e^{-\lambda_{ls} T_{ls}(x)}$$

where $M_{ls}(x)$ is the number of occurrence of events of type l while s is active in the event sequence x, and $T_{ls}(x)$ is the total duration while s was active for event type l.

In practice, the learning of a PCIM is divided into two processes: the *local* structure learning process that could be done using a decision tree to alternate between active states and define adequate mapping from the data to the states set similarly to [CHM97]; and the *parameters* learning process that uses statistical estimates depending on the type of the model and knowing the structure. Commonly, a greedy search is used where for a fixed structure (for instance a given decision tree and a given mapping for states) the parameters are calculated, and the structure is continuously modified until we cannot find a better model based on a selection criterion or a certain "gain".

The theory of CIMs is a root to many continuous time formalisms. It is an expressive tool that can help build graphical models such as Poisson networks and GEMs, that will be detailed later on. Moreover, PCIMs favor the discretization of states instead of the discretization of time, creating a set of constant parameters associated with each "scenario" in time and history of past events in which an event can occur. In the following, we define a family of graphical models that is directly used in the rest of the chapter.

Graphical event models

Definition 17. A *Graphical Event Model (GEM)* is a tupple $\mathcal{G} = ((\mathcal{L}, E), \theta)$ where $(\mathcal{L}, E)$ is a directed graph and θ a set of parameters. A GEM can represent event streams of the type x as defined previously, as well as the dependencies between the different labels (or events) in time.

In this case, the likelihood of the data knowing the graph and its parameters is written as:

$$p(x_{t^*} \mid t^*) = \prod_{i=1}^{|x_{t^*}|} \lambda_{l_i}(t_i \mid h_i) \prod_{i=1}^{|x_{t^*}|+1} e^{-\sum_{l \in \mathcal{L}} \int_{t_{i-1}}^{t_i} \lambda_l(\tau \mid h_i) d\tau}, \tag{4.1}$$

with $h_{|x_{t^*}|+1}$ the entire history of the event stream, including x_{t^*}.

In figure 4.2, an example of a GEM is shown with four labels (nodes) A, B, C and D, each with its corresponding parameter (conditional intensity function). The structure of a GEM (directed graph) has no constraints over cycles or loops, giving it a superior degree of expressive power compared to other graphical models based on DAGs, like Dynamic Bayesian Networks for instance, that cannot directly allow cycles or loops in a single time-slice.

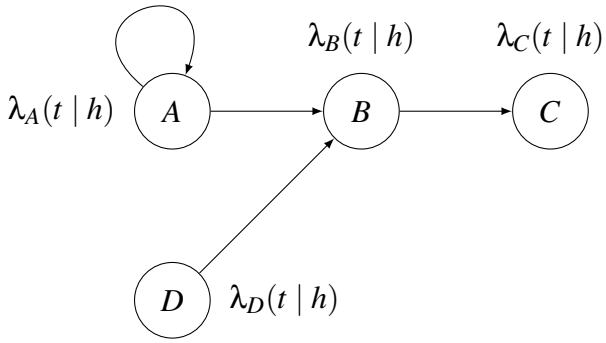


Figure 4.2 – Example of a 4 labels GEM

In this formalism, one should note that the conditional intensity functions $\lambda_l(t \mid h)$ are not *piecewise-constant*, which means that they do not take constant values for a certain period of time so the models are a family of CIMs and not PCIMs.

In this work we are only interested in *Markov* m.p.p.s with respect to a certain GEM. The conditional intensity functions $\lambda_l(t \mid h)$ in this case satisfy the following property for all t and h:

$$\lambda_l(t \mid h) = \lambda_l(t \mid [h]_{Pa(l)})$$

where $Pa(l)$ are the parents of l in $\mathcal{G}$. This means that the conditional intensity of a certain label l at time t only depends on the history of the parents of l and not the entire history of the process.

The dependencies between the events can be easily depicted on the graph of the GEM. In

the graph of figure 4.2 for instance, one can identify that label B has two parents A and D, hence $\lambda_B(t \mid h)$ depends on the past history of A and D. However, label D does not have any parents and subsequently has a constant rate $\lambda_D(t \mid h) = \lambda_D$ of occurrence.
A fully connected GEM can represent any m.p.p. with labels in $\mathcal{L}$ [GM16].
We now introduce Timescale GEMs, a subfamily of GEMs that is more granular and can be more easily used in practice.

Timescale graphical event models

A Timescale GEM (TGEM) is a GEM where each dependency between two events (edge in the graph) is defined for a given finite *timescale* which specifies the temporal horizon and the granularity of the dependency represented by that edge. Formally, a timescale is a set T of half-open intervals $(a,b]$ (with $a \geq 0$ and $b > a$) that form a partition of some interval $(c,t_h]$ (with $c \geq 0$), where t_h is the highest value of T and is called the *horizon* of the corresponding edge.

Definition 18. A TGEM $M = (\mathcal{G}, \mathcal{T})$ consists of a GEM $\mathcal{G} = ((\mathcal{L}, E), \theta)$ and a set of timescales $\mathcal{T} = T_{e_{(e \in E)}}$ corresponding to the edges E of the graph of $\mathcal{G}$.

We write $c_l(h,t)$ as the *parent count vector* of bounded counts (of occurrences) over the intervals in the timescales of the parents of l (in order to have bounded counts, a maximum threshold should be fixed). We provide an example to explicitly show parent count vectors and parameters (see Example 14).

The conditional intensity of a node (or a variable) labeled l in the GEM only depends on the history of the number of occurrences of its parents within the corresponding timescale. As a consequence, a given parent count vector yields a unique conditional intensity function for the corresponding parameter. The global conditional intensity functions are therefore piecewise-constant:

$$\lambda_l(t \mid h) = \{\lambda_{l,c_l(h,t)}\}_{c_l(h,t) \in C_l}.$$

We use C_l to denote the set of all possible parent count vectors of label l. For the following example (Figure 4.3), we consider that all TGEMs are bounded by 1 (making the parent count vectors binary), thus only the fact that a parent has occurred (or not) within the corresponding timescale is important and not the number of times a parent occurs.

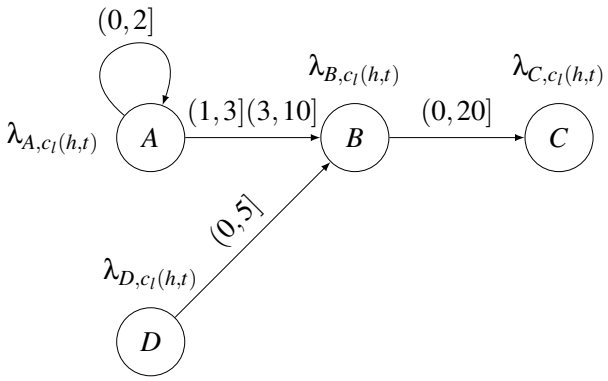


Figure 4.3 – Example of a 4 labels TGEM

Example 14. Consider the TGEM illustrated in Figure 4.3. We have $\mathcal{L} = \{A,B,C,D\}$, so we can list the different parent count vectors in C_l associated with each label: $C_A = \{0,1\}$, $C_B = \{0,1\} \times \{0,1\} \times \{0,1\}$, $C_C = \{0,1\}$ and $C_D = \varnothing$. Thus, for the event B for example, $c_B(h,t) = [0,1,1]$ means that there was no A in $[t-3,t-1)$, there was an A in $[t-10,t-3)$ and there was a D in $[t-5,t)$. Hence, the conditional intensity functions for the variable B are of the form: $\lambda_{B,000}$, $\lambda_{B,001}$, $\lambda_{B,010}$, $\lambda_{B,011}$, $\lambda_{B,100}$ $\lambda_{B,101}$, $\lambda_{B,110}$ and $\lambda_{B,111}$. The same applies to the rest of the variables, except the variable D that is independent of all other variables, so its conditional intensity function is written as λ_D. All conditional intensity functions are equal to constants making them piecewise-constant depending on the corresponding combination of parents.

In the case of TGEMs, the likelihood of the data in equation 4.1 is simplified and written as follows:

$$p(x_{t^*} \mid t^*) = \prod_{l \in \mathcal{L}} \prod_{j \in C_l} \lambda_{l,j}^{n_{t^*,l,j}(x_{t^*})} e^{-\lambda_{l,j} d_{t^* l,j}(x_{t^*})}, \tag{4.2}$$

where the sufficient statistics $n_{t^*,l,j}(x_{t^*})$ and $d_{t^* l,j}(x_{t^*})$ are the count of l-events and the durations, respectively, when the parent count vector was equal to j (a certain combination of parents). In [GM16], the sufficient statistics are formally written as:

$$n_{t^*,l,j}(x_{t^*}) = \sum_{i=1}^{|x_{t^*}|} \mathbf{1}(l_i = l)\mathbf{1}(c_l(h_i,t_i) = j)$$

$$d_{t^*,l,j}(x_{t^*}) = \sum_{i=1}^{|x_{t^*}|+1} \int_{t_{i-1}}^{t_i} \mathbf{1}(c_l(h_i,\tau) = j)d\tau$$

where $t_{|x_{t^*}|+1}$ represents the duration d between the last event occurence and the final time t^* in

the data.

Recursive timescale graphical event models

In practice, the learning process of GEMs is not an easy task, especially because they are not piecewise-constant intensity models, thus the learning of the conditional intensity functions can become hard (since they are not constants). The difficulty of learning GEMs, along with the necessity of have a "universal" Graphical Event Model that is easy to learn and to handle, led to the creation of a subclass of TGEMs called *Recursive Timescale Graphical Event Models* (RTGEMs).

Definition 19. The family of RTGEMs is defined recursively to be the finite closure of the empty model $\mathcal{M}_0 = ((\mathcal{L}, \{\}), \{\})$ under a set of allowed operators $O_F = \{add, split, extend\}$ (discussed in the following).

RTGEMs as described in [GM16] can universally approximate any smooth multivariate temporal point process. RTGEMs are learned recursively using the set of operators introduced above. More details on the learning procedure are given in the following.
Furthermore, it is proven in the works of [GM16] that using a greedy search algorithm to learn this class of models always shows structural and parametric consistency. In other words, the learned model converges in probability to the optimal model when the learning data size is increased, unlike the more general cases of TGEMs and GEMs for which, to the best of our knowledge, consistency has not been proven.

A finite consistent RTGEM learning procedure [GM16] is to do a forward greedy search (for model construction) followed by a backward greedy search (for model refinement), both based on model selection, and using data that is faithful. Bayesian Information Criterion (BIC) [S$^+$78] is a very general model criterion that has been adapted to select the "best" (or a "better") RTGEM when doing a greedy search, and is written as follows for a model M:

$$S_{t^*}(M) = log(p(x_{t^*} \mid t^*; M, \widehat{\lambda}_{t^*,l,j}(x_{t^*}))) - \sum_{l \in \mathcal{L}} \mid C_l \mid \cdot log(t^*),$$

where the left term of the difference is the likelihood of the data knowing the history, the model and the calculated maximum likelihood estimates of the λ functions (written $\widehat{\lambda}$). The right term represents the complexity of the model, with $\mathrm{Dim}(M) = \sum_{l \in \mathcal{L}} \mid C_l \mid$ as the dimension of an RTGEM, multiplied by the size of the sample used for learning to define the overall complexity.

Given a model M, the maximum likelihood estimate $\widehat{\lambda}$ of the λ parameters for M (knowing its structure) is as follows [GM16] :

$$\widehat{\lambda}_{t^*,l,j}(x_{t^*}) = \frac{n_{t^*,l,j}(x_{t^*})}{d_{t^*l,j}(x_{t^*})}$$

Now that RTGEMs are introduced, we explain how such models can be learned, sampled and compared.

4.3 Learning, sampling and comparing RTGEMs

The purpose of GEMs is to represent statistical dependencies between variables that one can observe on a data stream. However, the process of building a GEM that faithfully represents the statistical dependencies present in a given data stream can be arduous. RTGEMs have been introduced in [GM16] in order to solve this problem. By restricting the forms that the dependencies can take to timescales and the set of operations that can be done to build these timescales, the authors have shown that the learning time is drastically reduced. In the following, we present the set of operators that can be used in order to build an RTGEM and briefly explain how the learning process is performed. Sampling the model is also very important in our context in order to be able to perform SMC. We therefore explain next how RTGEMs can be sampled. Finally, we introduce one of our contributions: the definition of a distance measure that allows to compare two given RTGEMs.

4.3.1 Learning RTGEM

As explained in Sextion 4.2.2, the learning procedure defined in [GM16] consists of a forward greedy search for model construction followed by a backward greedy search for model refinement. Both the forward and the backward greedy search rely on a model selection criterion. As explained in Section 4.2.2, we usually use the Bayesian Information Criterion adapted to RTGEMs for this purpose.

The inital model consists of a graph with no edges whose vertices are the variables present in the given data stream. During the forward greedy search, the edges of the model (representing the dependencies between the variables) are progressively built by using a limited number of *forward elementary operators*. The set of forward elementary operators is the following: $O_F = \{add, split, extend\}$. The "add" operator adds a non-existing edge to a model and its

corresponding timescale $T = (0, c]$, with c a constant (also called horizon). The "split" operator splits one interval $(a, b]$ in the timescale of a chosen edge into two intervals $(a, \frac{a+b}{2}], (\frac{a+b}{2}, b]$. The extend operator extends the horizon of a chosen edge by adding the interval $(t_h, 2t_h]$, with t_h being the previous horizon.

After a number of forward operations (either a fixed number or when the improvement of the model selection criterion is sufficiently low), the forward search ends and a backward search starts. In this backward search, symmetric backward operators are used. For the sake of convenience, and since there are no further information in the literature about these operators, one can write $O_F^{-1} = \{reverse\_add, reverse\_split, reverse\_extend\}$ for these symmetric backward operators.

The "reverse_add" operator removes a chosen edge with only one interval in its timescale (to make it the exact inverse of the "add" operator). The "reverse_split" operator is used for merging two consecutive intervals in a timescale on a chosen edge that have been initially split. The "reverse_extend" operator removes the highest (the last) interval in a timescale on a chosen edge only if the upper bound of this interval was initially created by an extend.
Again, once a sufficient number of operators have been applied, or when the improvement of the model selection criterion is sufficiently low, the backward search ends and a definitive model is produced.

4.3.2 Sampling RTGEM

Once the model is learned, our aim is to verify if it satisfies given security properties. Since we use SMC for this verification, we need to produce samples of the learned RTGEMs. The sampling of such models is similar to the sampling of Poisson Networks [RGH05]. The needed "elementary" sampling techniques are demonstrated using the example in Figure 4.4 where different simple RTGEMs are shown. In Figure 4.4a the sampling is straightforward since there is only a single constant rate λ_A. In Figure 4.4b, the sampling of A is straightforward but the sampling of B will depend on the sampling of A. As a consequence, A must be sampled before B, then the active λ_B should be chosen respectively in regards to the occurrences of A along the timeline, and finally B can be sampled using the corresponding λ_B value in each section of the timeline.
Finally, in Figure 4.4c, things become more complicated: none of the variables can be sampled straightforwardly because of the cycle between A and B. Therefore, some sort of "competition" is

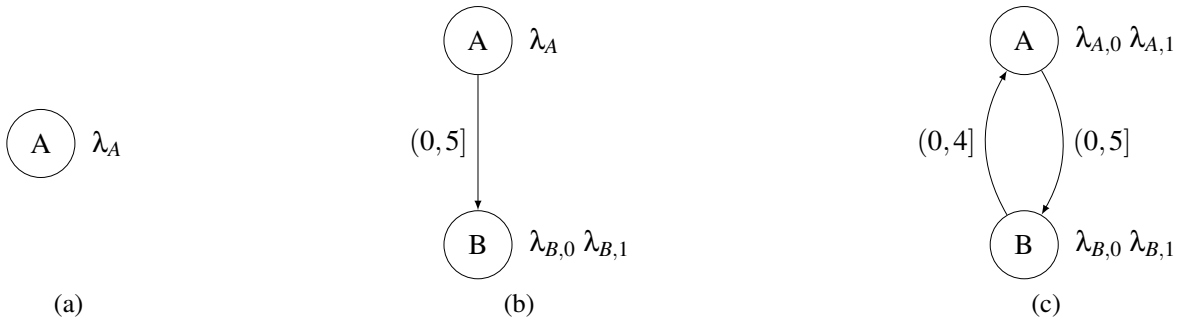


Figure 4.4 – Illustration of elementary sampling techniques for TGEMs

created between the variables and two time values τ_A and τ_B are sampled using the corresponding rates λ_{A,c_l} and λ_{B,c_l} (c_l is the corresponding parent configuration, $c_l = 0$ initially). The highest value (between τ_A and τ_B) is rejected because the one that comes before is supposed to change the corresponding conditional intensity function of the other (making the latter one wrongly sampled). Note that the corresponding rates are the conditional intensity functions that are active (with regards to the parents configuration) at the moment of the sampling. Afterwards, the lowest value (between τ_A and τ_B) is accepted if it is within the firing period. The firing period is the lowest interval on the timescale of the edge entering the corresponding node (for A it is between 0 and 5), because usually after that interval there is a switch of parents configuration and subsequently a switch in the conditional intensity function.

Using these elementary techniques, sampling can be generalized to a larger number of nodes, but two operations (illustrated in Figure 4.5) need to be performed beforehand.

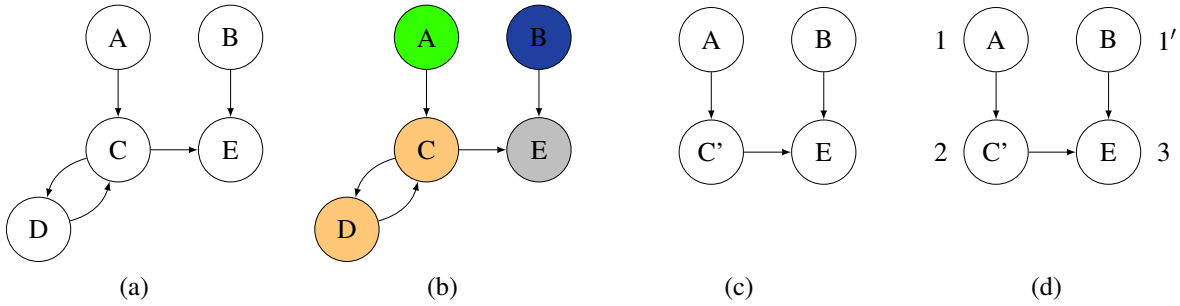


Figure 4.5 – Example from (a) to (d) showing how an RTGEM (we omitted the timescales for a better representation) can be transformed into ordered SCCs

The first operation is based on the concept of Strongly Connected Components (SCCs). A Strongly Connected Component (SCC) in graph theory [CLRS01] is a directed subgraph where

there exists a path between every single pair of nodes. The first operation is illustrated in Figure 4.5b where strongly connected components (nodes of the same color) are brought together forming a new "node" locally and the directed graph is transformed to a DAG at the end of the operation. The second operation (illustrated in Figure 4.5d) is topological ordering [CLRS01], and it defines an order that can be used for sampling the new nodes of the DAG using the three elementary operations. Note that the topological order is not unique, for instance 1 and $1'$ could be sampled in random order. A detailed description of the sampling procedure introduced above is given in [Ant20].

We now introduce one of our theoretical contributions: a notion of distance that allows to compare two given RTGEMs.

4.3.3 Distance between RTGEMs

To the best of our knowledge, there is no existing metric between RTGEMs. The most intuitive distance measure that comes to mind is the minimal number of operations needed to move from one RTGEM to another since they are built using a recursive procedure. However, such a distance is not accurate in our context because not all operators add (or remove) the same information every time.

In the literature, the popular Hamming distance [Ham50] has been adapted for some probabilistic graphical models such as Bayesian networks [TBA06]. In the following, we propose an extension of the Structural Hamming Distance (SHD), adapted to RTGEMs, where we evaluate the amount of differing information on two different edges.

A timescale in an RTGEM can be represented by a vector $v = [0, a, b, c, \ldots]$ containing the successive endpoints values. Given two RTGEMs with the same set of labels $\mathcal{L}$, $G_1 = ((\mathcal{L}, E_1), \theta_1)$ and $G_2 = ((\mathcal{L}, E_2), \theta_2)$ and an edge $e \in E_1 \cap E_2$, we write v_1^e and v_2^e for the values of the respective timescales of e in G_1 and G_2. We then write $v_{id}^e = |v_1^e \cap v_2^e|$, for the number of identical endpoints in the two vectors; and $v_{nid}^e = |v_1^e \setminus v_2^e| + |v_2^e \setminus v_1^e|$, for the number of endpoints that are not identical in the two vectors. This allows to define an elementary distance between the two respective timescales of the edge e as follows:

$$d(v_1^e, v_2^e) = \frac{v_{nid}^e}{v_{nid}^e + v_{id}^e} \tag{4.3}$$

Using these notations, we define the *Structural Hamming Distance (SHD)* between G_1 and G_2 as follows:

$$\text{SHD}(G_1, G_2) = |(E_1 \setminus E_2) \cup (E_2 \setminus E_1)| + \sum_{e \in E_1 \cap E_2} d(v_1^e, v_2^e) \tag{4.4}$$

Each edge that is present in one RTGEM and not in the other therefore adds 1 to the SHD, while each edge that is present in both adds the elementary distance between its respective timescales. One can show that the SHD is indeed a distance metric (*i.e.*, satisfies the distance axioms). A proof is given in [Ant20].

While this distance allows to compare both the structure and the timescales of two RTGEMs using the same labels, it suffers from a "scaling" disadvantage when it comes to comparing quantitative information. For instance, consider three vectors $v_1^e = [0,1,2]$, $v_2^e = [0,1.01,1.98]$ and $v_3^e = [0,5,10]$, representing the endpoints of three different timescales to compare using the distance measure. By computing the elementary distances between (v_1^e, v_2^e) and (v_1^e, v_3^e) with respect to equation 4.3, we notice that they are equal although v_1^e and v_2^e cover almost the same timescale compared to v_3^e that covers a different timescale. We have therefore proposed an extension to this SHD where the relative quantitative differences inside the timescales are taken into account in order to give a fairer distance measure.

The idea is to find matches (if existing) between the endpoints of the two timescales based on the mutual minimal absolute difference. In other words, consider two vectors v_1^e with size l and v_2^e with size k. A match is a pair $(v_{1_i}^e, v_{2_j}^e)$ such that the closest element from $v_{1_i}^e \in v_1^e$ in v_2^e is $v_{2_j}^e$, and the closest element from $v_{2_j}^e \in v_2^e$ in v_1^e is the same $v_{1_i}^e$. Formally, the closest element from $v_{1_i}^e \in v_1^e$ in v_2^e is written

$$\mathbf{cl}(v_{1_i}^e, v_2^e) = argmin_{v_{2_p}^e}(|v_{1_i}^e - v_{2_p}^e|).$$

Using this notion, the set of matches for a given edge $e \in E_1 \cap E_2$ is written V_{id}^e and defined as follows:

$$V_{id}^e = \{(v_{1_i}^e, v_{2_j}^e) \in v_1^e \times v_2^e : \mathbf{cl}(v_{1_i}^e, v_2^e) = v_{2_j}^e \wedge \mathbf{cl}(v_{2_j}^e, v_1^e) = v_{1_i}^e\}$$

We write $V_{id,1}^e$ for the projection of V_{id}^e on v_1^e, *i.e.*, $V_{id,1}^e = \{v_{1,i}^e, \exists j, (v_{1,i}^e, v_{2,j}^e) \in V_{id}^e\}$. Similarly, we write $V_{id,2}^e$ for the projection of V_{id}^e on v_2^e. The set of unmatched endpoints can now be defined as:

$$V_{nid}^e = (v_1^e \setminus V_{id,1}^e) \cup (v_2^e \setminus V_{id,2}^e).$$

Using these sets, we enlarge the set of endpoints that are considered identical in the elementary distance between two timescales. We now write $\mathbf{v}_{id}^e = |V_{id}^e|$ for the number of proximally matched

endpoints and $\mathbf{v}^e_{nid} = |V^e_{nid}|$ for the number of unmatched enpoints. The elementary distance between v^e_1 and v^e_2 can now be written

$$d^*(v^e_1, v^e_2) = \frac{1}{\mathbf{v}^e_{nid} + \mathbf{v}^e_{id}} \left(\sum_{(v^e_{1_i}, v^e_{2_j}) \in V^e_{id} \setminus (0,0)} \frac{|v^e_{1_i} - v^e_{2_j}|}{min(v^e_{1_i}, v^e_{2_j})} \right) + \frac{\mathbf{v}^e_{nid}}{\mathbf{v}^e_{nid} + \mathbf{v}^e_{id}}.$$

Remark that for each pair of matched endpoints, the relative difference (scaled by its minimum) is taken into account in order to penalize the fact that they may not be perfectly identical. Obviously, this factor is smaller than 1 (otherwise either $v^e_{1_i}$ or $v^e_{2_j}$ would be closer to 0 than to its counterpart, which would imply that $(v^e_{1_i}, v^e_{2_j}) \notin V^e_{id}$). As a consequence, pairs of imperfectly matching endpoints are less penalized than endpoints that have no match, but still more penalized than pairs of identical endpoints.
Finally, if there are no detected imperfect matches (*i.e.*, $V^e_{id} = \{(v^e_{1,i}, v^e_{2,j}) : v^e_{1,i} = v^e_{2,j}\}$), then we have $d^*(v^e_1, v^e_2) = d(v^e_1, v^e_2)$.

The *proximal distance measure*, that will be used in the rest of the chapter, takes into account this similarity between endpoints and is defined similarly to the SHD but using the elementary distance d^* defined above instead of d:

$$\mathrm{SHD}^*(G_1, G_2) = |(E_1 \setminus E_2) \cup (E_2 \setminus E_1)| + \sum_{e \in E_1 \cap E_2} d^*(v^e_1, v^e_2). \tag{4.5}$$

Although this is not a problem in our context, we remark that the proximal distance defined above is not, strictly speaking, a distance metric but only a semimetric (it does not satisfy the triangular inequality). Counter examples are given in [Ant20]. Because it suits our purpose, we will nevertheless call it a *distance* in the following, but this is an abuse of language.

4.4 Experimentations

As explained before, the aim of this work to measure how secure a system, which we can only observe through pre-generated outputs, is w.r.t. a given qualitative security property of the form $\mathbb{P}(\varphi \mid M) > c$ where φ is a linear property and c is a constant [2] To this aim, we have developped a strategy that can be summed up by Algorithm 1.
The only inputs we have in this context are the query φ and some data $\mathcal{D}$. The algorithm is in three

2. In the following, φ is called a *query* and the term *security property* refers to $\mathbb{P}(\varphi \mid M) > c$.

Algorithm 1 Proposed Strategy

input: $\mathcal{D}, \varphi$
output: M^*, Δ
1: $M^o = \underset{M \in \text{RTGEM}}{\text{argmax}}\ \mathbb{P}(\mathcal{D} \mid M)$
2: If $\mathbb{P}(\varphi \mid M^o) > c$ Then
3: Return$(M^o, 0)$
4: Else
5: $M^* = \text{find}\{M \in \mathcal{N}(M^o), P(\varphi \mid M) > c\}$
6: $\Delta = \text{Distance}(M^o, M^*)$
7: Return(M^*, Δ)

main steps. The first step (line 1) consists in learning a model M^o that best represents the data $\mathcal{D}$. This model is called the *fittest model*. The second step (line 2) consists in verifying whether the obtained model M^o satisfies the security property. If this is the case, then the algorithm ends. Otherwise, the third step (lines 5-6) consists in searching the neighbourhood of M^o for a model M^* that satisfies the security property and, if one is found, compute the distance between M^o and M^*. If none is found then the empty model $\perp$ is returned and the distance is considered infinite. Remark that the algorithm presented above is generic, *i.e.*, it does not depend on the choice of the modeling formalism, verification technique or distance notion. In the following, we present experiments performed for each step of this algorithm using the techniques and notions presented in the previous sections, but other techniques and notions could be used instead.

To the best of our knowledge, there is no existing tool in the literature implementing the learning, sampling or verification of RTGEM. We have therefore developed a branch of the PILGRIM library [3] dedicated to the manipulation of RTGEM, which we use in the rest of the chapter. All the experiments presented hereafter have been performed on a Windows 10 Enterprise i5 laptop, with a 2.3 GHz CPU and 8GB of RAM.

4.4.1 RTGEM learning performance

We start with an experiment designed to evaluate the performance of the learning algorithm for RTGEM presented in Section 4.3.1. To this purpose, we have considered a benchmark consisting in several random RTGEM (which we call references) generated with a prefixed complexity. From these RTGEM, we generate data samples of different length and use the learning technique of Section 4.3.1 on those samples to learn new RTGEMs. We then measure

3. `http://www.pilgrim.univ-nantes.fr/`

the proximal distance between the learned RTGEMs and their corresponding reference RTGEM to evaluate the accuracy of the learning algorithm w.r.t. the size of the input data, and compute the mean learning times to evaluate the performance of the learning algorithm.
For this experiment, we used RTGEMs of different complexity (number of vertices and timescales), written $|C_l|$. For each RTGEM complexity, we generated 50 random models [4] from which we sampled data of varying length, written t^*. We then use each sampled data to perform the RTGEM learning algorithm.
The mean values of learning times are presented in Table 4.6, where we can see the variations in learning duration depending on the size of the sample and the complexity of the RTGEM. Note that we fix the same number of nodes (labels) for every batch of graphs for the sake of convenience. In addition, the parameters λ are drawn randomly from a set of parameters, whose values are fixed beforehand in a way not to bias the tests (not very small nor very large values).

Dim(M) / t^*	$\|C_l\| = 8$	$\|C_l\| = 12$	$\|C_l\| = 32$	$\|C_l\| = 44$	$\|C_l\| = 80$
500 t.u.	0.3 sec	0.54 sec	1.72 sec	2.24 sec	4.21 sec
1000 t.u.	0.62 sec	0.94 sec	4.27 sec	4.74 sec	8.57 sec
10 000 t.u.	1.72 sec	6 sec	42 sec	1 min 6 sec	1 min 58 sec
20 000 t.u.	5 sec	14.35 sec	1 min 25 sec	2 min 32 sec	4 min 40 sec
25 000 t.u.	7.23 sec	18.97 sec	2 min 41 sec	5 min 38 sec	11 min 20 sec
50 000 t.u.	41.67 sec	58.51 sec	6 min 38 sec	13 min 37 sec	24 min 11 sec

Table 4.6 – Variations in learning time depending on the size of the sample and the complexity of the model

From these results we can conclude that the bigger the complexity and the sample size, the more time consuming the learning is, which is not surprising. Still, we can see that learning of RTGEM can be performed on a standard computer with acceptable performance even with large data representing large RTGEM.

In Figure 4.7, we show plots representing the variation of the mean of the normalized distances between the learned and original RTGEM in function of the size of the sample for different complexities. The standard deviation is indicated by the shadowed curve around the plot for each complexity. The normalized distance is the $SHD^*(reference, learned)$ divided by the maximum possible distance between two graphs of the given complexity, *i.e.*, the square of the total number

4. Details on the random generation process are left out of this document but can be found in [Ant20].

of nodes $|\mathcal{L}^2|$. We write $D(reference, learned) = \frac{SHD^*(reference,learned)}{|\mathcal{L}^2|}$.

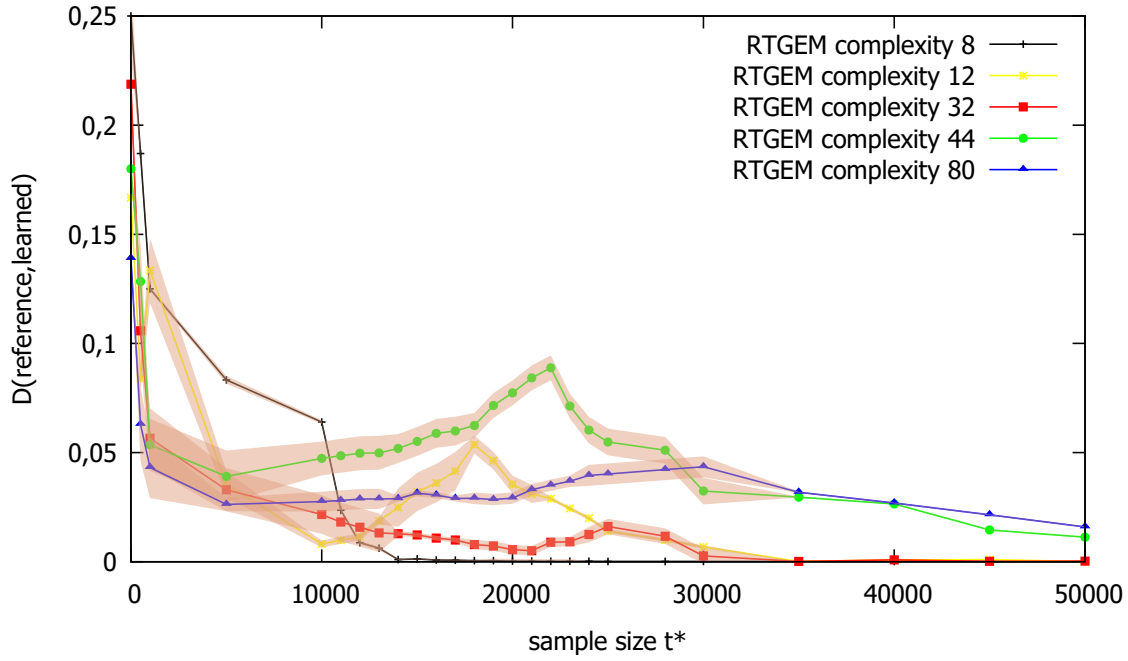


Figure 4.7 – Mean normalized distances between learned RTGEMs with different complexities and their corresponding references, in function of the sample size t^*

The main observations that can be made based on this plot is the common trend between all graphs, from every complexity, to get closer to the reference model as the sample size increases. Moreover, we can see that the more complex the reference model is the harder it is to learn a model that is close to it (more data is needed). However, we can observe (mainly between $t^* = 10000$ and $t^* = 25000$) some fluctuations that are due to the learning of inaccurate models. We conjecture that this is happening when there is enough data to select an RTGEM that models inaccurate behaviors, but still not enough data to learn the accurate dependencies. Some explanations and more detailed experiments are given in [Ant20].

4.4.2 Statistical model checking RTGEM

We now move to an evaluation of the second step of Algorithm 1: verifying whether the fittest model satisfies the given security property. Note that sampling performance is an important part of SMC. In [Ant20], we discuss and evaluate an optimization of the sampling strategy that is

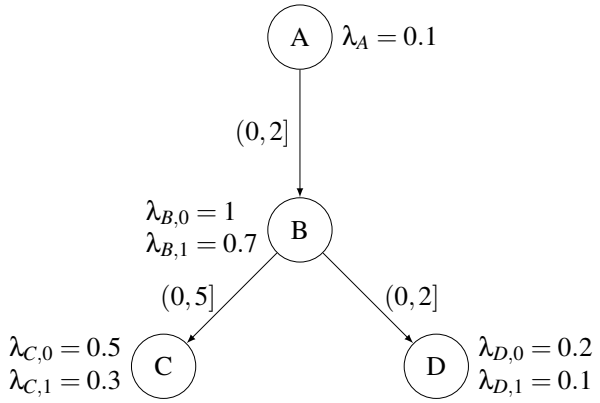


Figure 4.8 – The RTGEM M_1 on which we apply statistical model checking to verify the query φ_1

used in the following experiments, which we call early stopping.
We start with the verification of a simple property in Example 15 by using a quantitative approach (with Monte-Carlo simulations) and a qualitative approach using the Sequential Probability Ratio Test (SPRT).

Example 15. Consider the graph of the RTGEM M_1 in Figure 4.8 and the query $\varphi_1 = \Box^{100}(B[0,5])$, meaning that event B must occur at least every 5 time units during a 100 time units period. We want the model to satisfy the security property $\mathbb{P}(\varphi_1 \mid M_1) > 0.80$. For the sake of coherence we always use the name security query to refer to the linear property φ, and the name security property for the qualitative inequality $\mathbb{P}(\varphi \mid M) > c$ with M the model and c a threshold ($c \in [0,1]$).

The first experiments targets the *quantitative* approach based on Monte-Carlo simulations first intoduced in Section 3.3.1. A precision (δ) and an error rate (ε) must be fixed beforehand. The number of simulations (N) is also computed beforehand via the formula:

$$N = \frac{ln(2) - ln(\varepsilon)}{2\delta^2}$$

This technique allows to compute an estimate $\widehat{p}(\varphi_1|M_1)$ to $\mathbb{P}(\varphi_1 \mid M_1)$ with guaranteed precision. Table 4.9 shows the results for this quantitative test. Unfortunately, to the best of our knowledge, there are no exact techniques that are adapted to this kind of formalism in order to compute the real probability $\mathbb{P}(\varphi_1 \mid M_1)$ and compare it with the obtained estimations.
Remark that although the property we are verifying is qualitative, it is also possible to apply a quantitative approach, have a numerical estimate (with a certain precision) and afterwards compare the estimate with the desired threshold.

	δ	ε	N	Duration	$\widehat{p}(\varphi_1 \mid M_1)$
Test 1	0.02	0.01	6622	5 min 54 sec	0.9177 ($>$ 0.80 property satisfied)
Test 2	0.008	0.005	46808	52 min 18 sec	0.92179 ($>$ 0.80 property satisfied)
Test 3	0.005	0.003	130044	2 hr 8 min	0.92182 ($>$ 0.80 property satisfied)

Table 4.9 – Results for quantitative statistical model checking of query φ_1 on M_1 with Monte-Carlo simulations

The results for this approach confirm that, as expected, for a high precision (δ) and a low error rate (ε) the number of simulations becomes high.

The second experiment targets the *qualitative* SPRT approach. In this context, we use a centered indifference region of size δ, centered in 0.8 (the target value) and strength parameters (α, β) (see Section 4.2.1). Contrary to the quantitative approach, the required number of simulations is not fixed beforehand, but rather determined dynamically, depending on the outcome of each simulation. This technique computes a decision about the security property $\mathbb{P}(\varphi_1 \mid M_1) > 0.8$. Table 4.10 shows the results for this qualitative test.

	δ	α	β	m	Duration	$\mathbb{P}(\varphi_1 \mid M_1) > 80\%$
Test 1	0.02	0.01	0.01	167	8.18 sec	Property satisfied
Test 2	0.008	0.005	0.005	275	16.21 sec	Property satisfied
Test 3	0.005	0.003	0.003	344	41.29 sec	Property satisfied

Table 4.10 – Different results for qualitative statistical model checking of query φ_1 on model M_1 with Sequential Probability Ratio Test.

Remark that the comparison of the performance of the quantitative and qualitative approach is situational [5]. Their relative performance can vary depending on the model and the property, although it is common knowledge that the qualitative approach is usually orders of magnitude faster *if the estimated probability is not too close to the fixed threshold*. Since we do not know beforehand whether this condition is satisfied, we will use both techniques in parallel in the following experiments, allowing the quantitative technique to conclude when the qualitative technique does not converge fast enough.

5. Further experiments can be found in [Ant20]

4.4.3 Testing the proposed strategy

We now propose our main experiment, that applies our whole strategy (Algorithm 1 to three different sets of synthetic data. We start with a description of the experimental context and then present the results for each attempt.

Experimental protocol

The aim of this experiment is to perform a proof-of-concept for our proposed strategy (given in Algorithm 1). We will therefore consider three cases: (1) when the learned model satisfies the given security property (case OK), (2) when the learned model does not satisfy the security property, but is close to satisfying it (case KO_1), and (3) when the learned model is far from satisfying the security property (case KO_2).
Our pipeline is shown in Figure 4.11, where $xxx \in \{OK, KO_1, KO_2\}$ and one can observe that we follow the strategy proposed in Algorithm 1. In the OK case, we use data generated from an RTGEM that satisfies the security property, while in KO_1 and KO_2, we use data generated from models that do not satisfy the security property. The details of the models will be given further on.

Remark. A part of our strategy relies on the exploration of the neighborhood of the fittest model M^o. For this experiment, we have chosen a notion of neighborhood based on the number of forward operators applied on an RTGEM. Moreover, we have chosen to perform a Random Walk on this neighborhood. In other words, we apply a limited number of random forward operators on the fittest model M^o in order to produce candidate models M^*. The maximum number of forward operators varies for each case (irrelevant for (OK), 25 for (KO_1) and 100 for (KO_2)) and have been set according to preliminary experiments.
In order to optimize the exploration procedure, we do not choose randomly from the whole set of possible operators, but rather from a subset of those operators that might have a more significant impact on the satisfaction of the property. For the sake of completeness, if no model is found when the whole subset of operators has been sampled, we extend our search to the remaining ones. This strategy is discussed in further details in [Ant20].

Experimental results for the proposed strategy

The synthetic data sets used for learning the models contain event streams with labels "Login", "Logout", "Check account", "Recharge" and "Transfer money". These sets were gen-

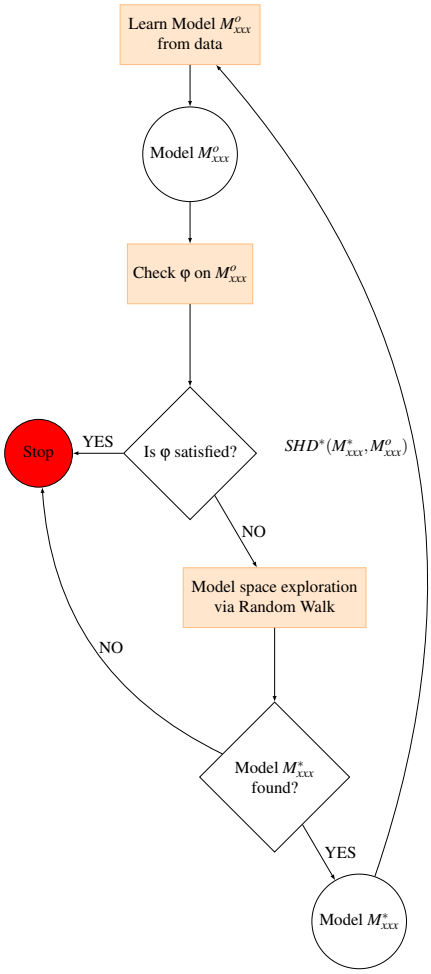


Figure 4.11 – An illustration of the experimental protocol that is conducted in this section

erated from RTGEMs having different structures and parameters describing different behaviors on a prepaid card online service. The security query that we use in this experiment is: $\varphi = \square^{1000}(\text{Transfer Money} \Rightarrow \text{Recharge}_{(0,20]} \vee \text{Check account}_{(0,5]})$, i.e. each time a transfer is performed, the user must have either recharged his/her account in the previous 20 time units, or checked his/her balance in the previous 5 time units. We want the model to satisfy the security property: $\mathbb{P}(\varphi \mid M) > 0.8$.

For the experiments, the statistical model checking techniques that we use are the qualitative Sequential Probability Ratio test and the quantitative estimation based on Monte-Carlo simulations. These techniques are used in parallel as previously explained. The SMC algorithms are calibrated as shown in Table 4.12. Remark that the maximum number of simulations that we do (for each model) is determined by the number of simulations corresponding to the quantitative

technique, in the case where SPRT does not converge faster to a result.

	α	β	δ	ε	Number of simulations
SPRT	0.05	0.05	0.01	-	Unknown beforehand
Monte-Carlo simulations	-	-	0.01	0.05	18444

Table 4.12 – Calibration of the SMC techniques for the following experiments

First case: (OK)**.** In the first test, we run our algorithm on the corresponding set of data to learn the model M^o_{OK}. The obtained model is shown in Figure 4.13. For the sake of simplicity we do not represent the parameters of the label "Logout" for all of the remaining figures, since it is not a target node with regards to φ.

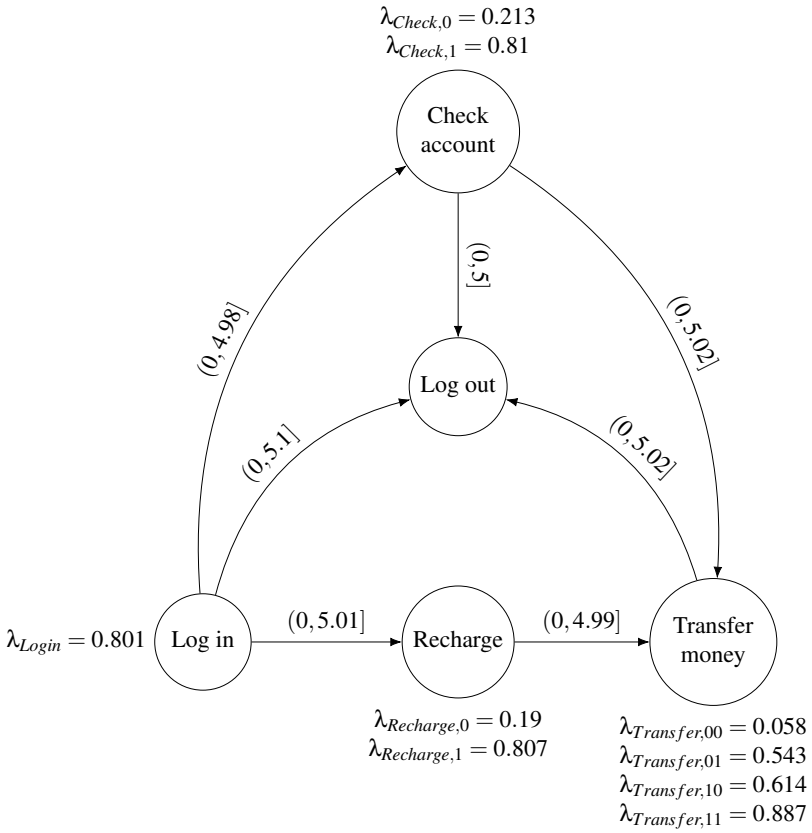


Figure 4.13 – A representation of the learned model M^o_{OK} from the set of "secure behavior" data

In Table 4.14, we show the obtained results after the completion of the first test. The formal verification with SMC is repeated twenty times in order to establish mean values for the required

duration and number of simulations before acquiring a result. We write $\widehat{t}$ for the mean duration and $\widehat{m}$ for the mean number of simulations over twenty executions for all remaining tests.

	$\mathbb{P}(\varphi \mid M) > 0.8$	$\widehat{t}$	$\widehat{m}$
M^o_{OK}	Property satisfied	5 sec ($\pm$0.43 sec)	277 simulations ($\pm$49 simulations)
M^*_{OK}	-	-	-

Table 4.14 – Results of the first test on M^o_{OK}

The learned model M^o_{OK}, that best represents the data, verifies the security property in this case. Hence, no further exploration is made and the test stops.

Second test: (KO_1)**.** For the second test, we run our algorithm on the corresponding set of data to learn the model $M^o_{KO_1}$. The obtained model is shown in Figure 4.15. The model $M^o_{KO_1}$ does not satisfy the security property. Hence, the space exploration technique is repeated twenty separate times in this test, in order to check whether every time we find a model that satisfies the property. Similarly, we check whether it is the same proximal secure model that is found every time or if there are different possible ones.

The space exploration is executed on the learned model $M^o_{KO_1}$ and only one proximal secure model ($M^*_{KO_1}$) is found in its neighborhood (see Figure 4.16). As previously mentioned, we use the Random Walk technique to explore the neighborhood of the learned model. The maximum number of allowed modifications (chosen operators) in this test is set to 25. Remark that the proximal secure model does not make sense w.r.t. the real-life application. Indeed, our aim is only to find a mathematical object, a combination of parameters, that are close to the fittest model $M^o_{KO_1}$ and provide samples of behaviors which are "secure" with regards to the property.

In Table 4.17, we show some results of the algorithm for this test. We write $\widehat{o}$ for the average number of applied modifications before finding a proximal secure model, $\widehat{\tau}$ for the average duration of the space exploration process (including the formal verification at each step), and n for the number of distinct proximal secure models found. The success rate is the number of times we found a proximal secure model over the total number of executed Random Walks. We use the same notations for the remaining test.

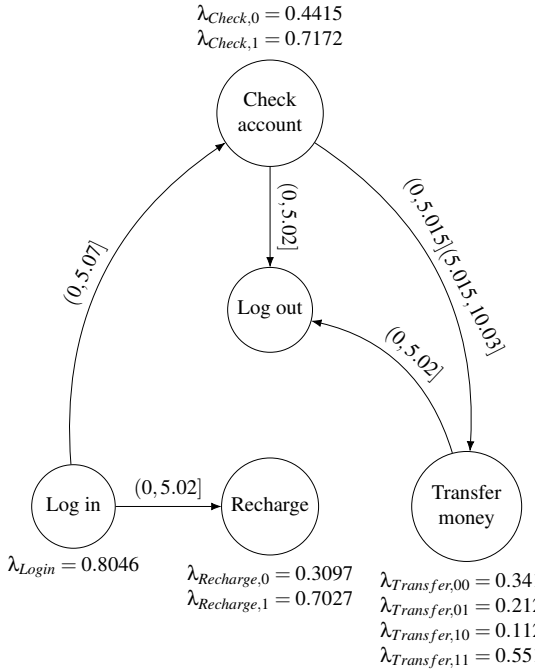


Figure 4.15 – A representation of the learned model $M^o_{KO_1}$ from the corresponding set of "not secure" data

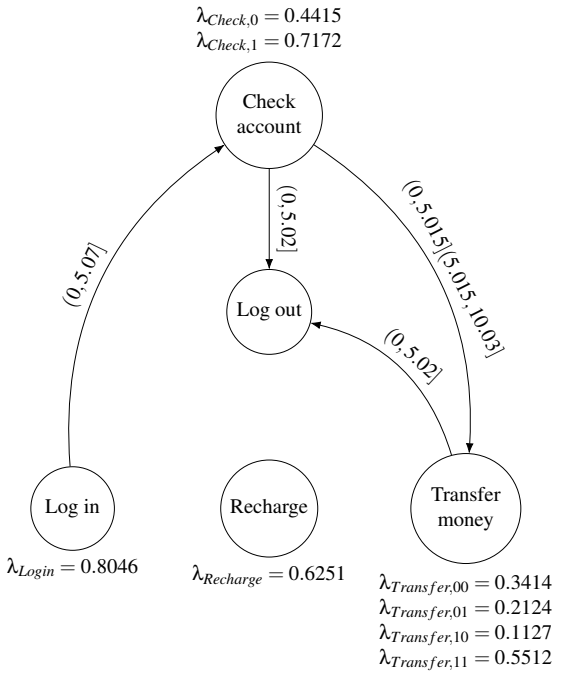


Figure 4.16 – A representation of the proximal model $M^*_{KO_1}$ which satisfies security property in the neighborhood of $M^o_{KO_1}$

	$\widehat{o}$	$\widehat{\tau}$	n	Success Rate	$SHD^*(M^*_{KO_1}, M^o_{KO_1})$
$M^o_{KO_1}$	12 operators ($\pm$2 operators)	13 min 52 sec ($\pm$30 sec)	1	20/20	1

Table 4.17 – Performances of the algorithm applied on the set of data KO_1

The last results for this test are shown in Table 4.18 after the completion of the experiment. The formal verification with SMC is repeated twenty times for both $M^o_{KO_1}$ and $M^*_{KO_1}$ in order to establish mean values for the required duration and number of simulations before converging on a result. Note that the SMC was executed only one time for all the intermediate models obtained at each step of the exploration process while constructing $M^*_{KO_1}$.

	$\mathbb{P}(\varphi \mid M) > 0.8$	$\widehat{t}$	$\widehat{m}$
$M^o_{KO_1}$	Property NOT satisfied	1 min 2 sec ($\pm$11 sec)	4018 simulations ($\pm$101 simulations)
$M^*_{KO_1}$	Property satisfied	2 min 17 sec ($\pm$25 sec)	5512 simulations ($\pm$122 simulations)

Table 4.18 – Results of the second test on $M^o_{KO_1}$

Third case: (KO_2)**.** For the third and final test, we run our algorithm on the corresponding set of data to learn the model $M^o_{KO_2}$. The obtained model is shown in Figure 4.19. The model $M^o_{KO_2}$ does not satisfy the security property. Hence, the space exploration technique is repeated twenty separate times in order to check whether every time we find a model that satisfies the property. Again, we check whether it is the same proximal secure model that is found every time or if there are different possible ones.

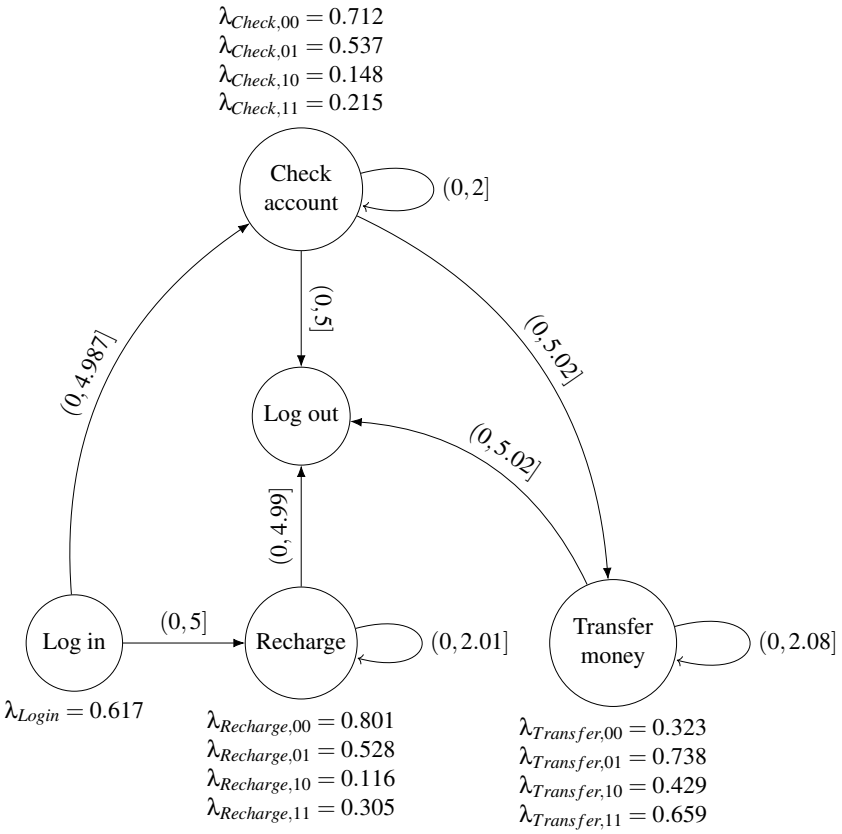


Figure 4.19 – A representation of the learned model $M^o_{KO_2}$ from the corresponding set of "not secure" data

The space exploration is executed on the learned model $M^o_{KO_2}$ and two proximal secure models ($M^*_{KO_{2_1}}$ and $M^*_{KO_{2_2}}$) are found in its neighborhood (see Figures 4.20 and 4.21). The maximum number of allowed modifications in the Random Walk (chosen operators) in this test is set to 100.

In Table 4.22, we show some results of the algorithm for this test. We use the same notations as in the previous case. The last column shows the distances for the two proximal models $M^*_{KO_1}$ and $M^*_{KO_2}$ in this order. Notice that the success rate is not perfect in this case: three times out of twenty we did not find a proximal secure model (failure rate of $3/20$). The proximal secure model

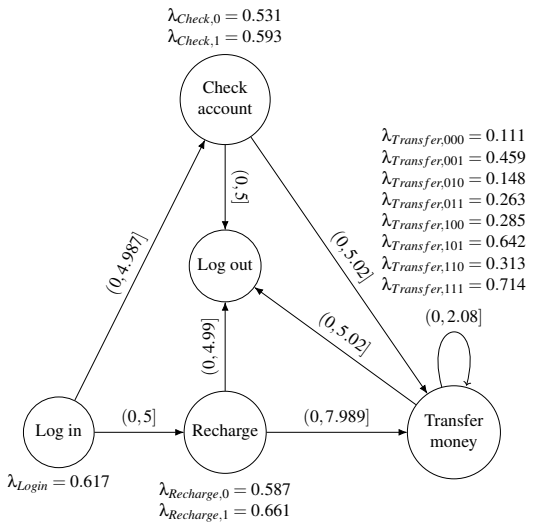


Figure 4.20 – A representation of $M^*_{KO_{2_1}}$ which satisfies the security property in the neighborhood of $M^o_{KO_2}$

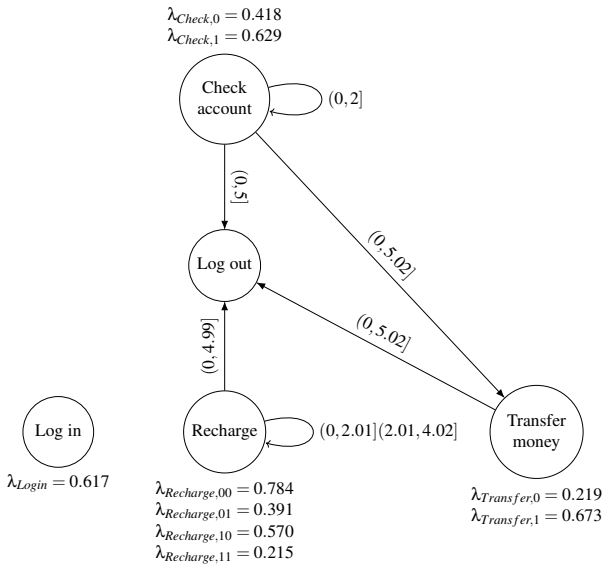


Figure 4.21 – A representation $M^*_{KO_{2_2}}$ which satisfies the security property in the neighborhood of $M^o_{KO_2}$

$M^*_{KO_{2_1}}$ (Figure 4.20) is found twelve times and the proximal secure model $M^*_{KO_{2_2}}$ (Figure 4.21) is found five times.

	$\widehat{o}$	$\widehat{\tau}$	n	Success Rate	$SHD^*(M^*_{KO_2}, M^o_{KO_2})$	
$M^o_{KO_2}$	88 operators ($\pm$8 operators)	1 hr 15 min ($\pm$9 min)	2	17/20	3	3.333

Table 4.22 – Performances of the algorithm applied on the set of data KO_2

The last results for this test are shown in Table 4.23. The formal verification with SMC is repeated twenty times for $M^o_{KO_2}$ and both $M^*_{KO_{2_1}}$ and $M^*_{KO_{2_2}}$ in order to establish mean values for the required duration and number of simulations before acquiring a result. Note that also in this test, the SMC was executed only one time for all the intermediate models obtained at each step of the exploration process.

	$\mathbb{P}(\varphi \mid M) > 0.8$	$\widehat{t}$	$\widehat{m}$
$M^o_{KO_2}$	Property NOT satisfied	3 min 22 sec ($\pm$11 sec)	2155 simulations ($\pm$69 simulations)
$M^*_{KO_{2_1}}$	Property satisfied	11 min 31 sec ($\pm$25 sec)	7512 simulations ($\pm$136 simulations)
$M^*_{KO_{2_2}}$	Property satisfied	7 min 19 sec ($\pm$25 sec)	4286 simulations ($\pm$114 simulations)

Table 4.23 – Results of the third test on $M^o_{KO_2}$

Discussion

These experiments have confirmed that our strategy for evaluating the security of a dataset w.r.t. a given security property is usable in practice for models of a medium complexity. However, we also remark from the performance evaluation on the KO_2 case that this strategy might not scale for models of a higher complexity.

Nevertheless, this experiment has allowed us to confirm that the main bottleneck of our strategy is the verification of the intermediate models, in particular due to the complexity of sampling data from RTGEMs. Indeed, we can see in Table 4.23 that the mean duration for verifying a secure model representative of the given data is around 10 minutes. Since an average of 88 operators were necessary to obtain the secure model, we can conclude that a consequent part of the total time (1h 15min) has been dedicated to running SMC on intermediate models.

Moreover, the experiment also confirms our intuition that finding a secure proximal model and computing the distance to the fittest model is more efficient when the input data corresponds to a model that is closest to satisfying the security property. Indeed, we can see that the whole procedure was performed in an average of 14 min in the KO_1 case while it took an average of $1h15min$ for the KO_2 case. In order to verify that KO_1 and KO_2 correspond to our intuition (*i.e.*, KO_1 closest to verifying $\mathbb{P}(\varphi \mid M^o_{KO_1}) > 0.8$, we have applied quantitative statistical model checking to the models used for generating both datasets. We obtained the following estimations: $\widehat{p}(\varphi \mid M^o_{KO_1}) = 0.728$ and $\widehat{p}(\varphi \mid M^o_{KO_2}) = 0.585$.

Of course, the above observations have to be taken lightly as they only reflect a few experiments, and would need a lot more for confirmation.

4.5 Conclusion

In this chapter, we have presented a strategy for measuring the security of a model observable through a given set of data w.r.t. a security property. To do that, we have used the RTGEM formalism along with a learning algorithm based on a forward and backward greedy search. Our models have been verified against the property using SMC techniques, and we have used a random walk technique to explore the neighbourhood of the learned model to find a secure one when this was needed.

Besides the immediate benefits of being able to rate the dangerousness of a given set of data, this work is, to the best of our knowledge, one of the first attempts at putting together learning and formal verification techniques *without using automata-based models as an in-between* (as is

done in [Leu06, Nei14]). This confirms that SMC is an appropriate verification technique even in the context of data-driven systems, where automata-based modeling formalisms are often inconvenient. This confirmation opens the way to many interesting perspectives, which we now present.

4.6 Perspectives

The technique presented in this chapter is a first attempt at verifying systems based on a given set of observations. It mainly serves as a proof of concept showing that SMC verification and RTGEM-based learning can be combined toward this end. However, there are many ways in which this technique can be improved, and many other domains to which it can be applied.

Potential improvements. We have observed that the bottleneck of our strategy is related to RTGEM sampling during the verification phase. In particular, we have observed that an improvement to sampling performance can be obtained by limiting the sampling to nodes related to the property under investigation (see [Ant20] for details). We conjecture that other techniques could be used to drastically enhance the sampling/verification performance such as taking advantage of partial samples drawn from unmodified parts of a previous RTGEM while exploring its neighborhood.
Similarly, we have chosen a random walk technique for exploring the neighborhood of the fittest model. While this technique has been slightly optimized by starting the random search with a reduced number of operators, we conjecture that one could take a better advantage of the given property in order to produce heuristics that would considerably reduce the required number of intermediate models that are verified before finding a secure proximal model.
Another potential improvement concerns the notion of distance between RTGEMS. Indeed, the notion of distance we have defined, which allows us to quantify the *security level* of a model, is not directly related to security. Although it provides a measure of how far a given set of data is from a secure model, this measure highly depends on the data itself and the complexity of the learned model. As a consequence, the absolute value obtained as an output for one set of data is difficult to compare to the one obtained for a different set. In the future, we plan to investigate other notions of distance that will allow us to compare the *security levels* obtained for different sets of data.

Other application domains. The technique developped in this chapter could be applied to other goals than security. In particular, RTGEM are a formalism that could be very useful in other

scientific domains, such as oceanography. In the context of the TARA Ocean community, in particular, a profusion of heterogenous timed data is collected and studied through statistical analysis. Using graphical models such as RTGEMs to represent this data would be particularly interesting as it would allow analyzing timing dependencies between the observed variables (which is mostly not done today). Moreover, the performances observed during the experiments performed in this chapter makes us confident that RTGEMs would be able to handle sufficiently large datasets to be usable in this context.

While we observed that sampling of RTGEMs is hard, it is nothing compared to the sampling of complex natural system models (*e.g.*, ocean biogeochemical models [AÉT$^+$15]). While some scientists have proposed using neural networks to simulate the outputs of climate models with high speed and precision [Hut20], the lack of understanding that still lies behind neural networks is a problem. An intermediate solution would be to use graphical models such as RTGEMs, which are well understood and give precise insight on the phenomena they represent, to enhance the speed of complex models sampling while preserving understandability of the model itself.

PERSPECTIVES

As can be seen from this document, my contributions in the past 10 years have followed two (somewhat separate) main lines of work. On the theoretical side, I have developed and studied modeling formalisms and verification techniques dedicated to the analysis of discrete and continuous parametric probabilistic systems. On the practical side, I have developed statistical methods and tools, and applied them to the learning and verification of complex systems. Unfortunately, in most cases, the modeling formalisms we develop on the theoretical side are not the ones we use on the practical side. Three main reasons stand out for this *gap* between theory and practice:

- The verification methods developed in the context of abstract modeling formalisms such as the ones presented in Chapters 1 and 2 are computationally expensive and often fail to scale up to models of industrial-size.
- The act of modeling a complex system requires deep expertise both on the system itself and on the chosen modeling formalism. Unfortunately, those who develop modeling formalisms and analysis techniques (who we will call model developers) rarely possess the domain expertise required to use these formalisms on concrete complex systems. On the other hand, those who use models in practice (who we will call modeling practitioners) rarely possess the theoretical expertise required to use abstract modeling formalisms.
- The existing barrier between natural/social sciences and formal sciences is such, that modeling formalisms are often not perfectly tailored to the use of modeling practitioners, and modeling practitioners are even rarely aware of the existence of such modeling formalisms.

While theoretical advances on modeling formalisms and verification techniques are without doubt important, I believe that the long-term aim of such research should be to provide better tools for the analysis of concrete systems. In order to do that, researchers in formal sciences need to make verification techniques more efficient and lessen the effort required from concrete modeling practitioners for using our modeling formalisms. During the past years, numerous interactions with Biologists and Oceanographers, in particular in the context of the Tara Ocean

GO-SEE research federation [6], have allowed me to refine the following research axes, which I believe could contribute to work toward this aim.

Enhancing parametric modeling and analysis for natural sciences

From the practical point of view, (parametric) statistical model checking stands out as a very promising technique for analyzing complex natural models. One of the main advantages of SMC in this context is the limited number of requirements on the chosen modeling formalism. Indeed, as we have shown in [REG+20], SMC can be applied with very light adaptations to any stochastic system modeled as a computer program, which is common in natural sciences. More requirements are needed in the context of parametric SMC, but the prototype tool we presented in [BAD+19] shows that Python programs where parameters are expressed in a certain form (*i.e.*, inheriting from predefined Python classes) can also be analyzed. Again, this can be generalized to other programming languages.

However, the types of parameters that can be taken into account in these models is not completely satisfactory. Indeed, the parametric SMC theory we developed is based on parametric Markov chains, where parameters can only range over transition probabilities. On the other hand, models of natural systems are often based on sets of parameterized differential equations, where the nature of the parameters is substantially different. Even when the differential equations are discretized, translating the equation parameters into transition probabilities is not an easy task. This is the main reason why [REG+20] is based on standard and not parametric SMC. Nevertheless, the only requirements about the applicability of (parametric) SMC are that (1) the model is executable, and (2) once the parameter values are fixed, the resulting model is purely stochastic. These requirements are met by models consisting of sets of parameterized differential equations, therefore there is no theoretical reason why (parametric) SMC methods could not be adapted to this context. In order to do so, we need to study: (i) new modeling formalisms where differential equation parameters can be taken into account as such, (ii) the form of the probability distribution on their traces, and (iii) how the results presented in Chapter 3 can be adapted to this new context.

6. The Tara Ocean GO-SEE (Global Oceans Systems Ecology & Evolution) research federation regroups 22 french and international research teams. In particular, the following organisms are involved in this federation: CNRS, CEA, Tara Ocean Foundation, Sorbonne Université, PSL, Inserm, ENS Paris, IRD, EPHE, Université d'Évry-Val-d'Essonne, Université Paris-Saclay, UPVD, AMU, Université de Toulon, École centrale de Nantes, Université de Nantes, UGA, EMBL and the Physics and Mathematics faculty of the University of Chile.

Developing new efficient abstraction techniques

Another limitation of (parametric) SMC is the number of simulations that are required in order to produce accurate results. Indeed, in the context of complex natural systems, producing simulations of a complete model can be computationally expensive. While a sufficient reduction of the number of required simulations is out of the picture [7], we believe that the solution lies in the development of abstraction methods that will reduce the computational complexity of performing simulations. Several such abstraction techniques can be envisioned. In particular, we plan to pursue the work initiated during my thesis on stochastic abstraction [BBB+12], a technique that allows to drastically reduce the size of the model in order to perform fast simulations. While the technique shows promise, the difficulty in this context is to be able to link the formal precision guaranteed by SMC algorithms with the loss of details resulting of the abstraction. Another lead, promoted in recent works [Hut20] is to use AI techniques based on neural networks in order to build a new model of the system whose outputs are statistically equivalent to those of the original one but that is orders of magnitude faster to simulate. However, the lack of understanding that still lies behind neural networks is a problem. Instead, we plan on developing such abstraction techniques based on other graphical probabilistic models such as the ones presented in Chapter 4, which yield more understandable models. Beyond that, we will use our expertise on *compositional* interface theories [Del10, CDL+10, CDL+11, DFH+12, DFLL13b, BDF+13, DKL+13, FLDL18] to study *compositional* versions of this technique, where only parts of the model are abstracted, in the spirit of what we did with stochastic abstraction [BBB+10a, BBB+12].

Building a verification platform for practitioners

The echos we have received from the Biological community for the results presented in [REG+20] have convinced us of the interest that a tool for automatic analysis and parameterization of natural systems expressed in standard programming languages would raise. Our intent in the near future is therefore to automatize the processes we have followed in [REG+20] in order to adapt the original model for SMC analysis and to build a platform that could be used by any modeling practitioner interested in such analysis, without further requirements. Our intent, as we did for the prototype tool presented in [BAD+19] is to propose two front-ends for this platform: one dedicated to the formal analysis of parametric "computer-science" models (*i.e.*, for the use of formal science researchers) where state-of-the-art verification techniques can be implemented

7. There exist statistical techniques for reducing this number of simulations, but they are limited and their effect is not sufficient compared to the computational complexity of producing a meaningful number of simulations.

and tested; and the second one dedicated to the concrete analysis/verification of models of natural systems (*i.e.*, for the use of modeling practitioners). Besides the development of new verification techniques, this platform raises many scientific and technical challenges on several topics such as parallelization, interpretation and rendering of dynamic partial results during a potentially very long verification, or visualization of meaningful information on high-dimensional sets of parameters.

These research axes are and will be the topics for multidisciplinary research projects (including Ph.D proposals) that will last several years and require support fundings. They are in line with the future priority research plan **Océans** of the CNRS that is at the moment being refined in the context of the corresponding Task Force (to which I participated).

BIBLIOGRAPHY

[A+15] Mohamed Abomhara et al. Cyber security and the internet of things: vulnerabilities, threats, intruders and attacks. *Journal of Cyber Security and Mobility*, 4(1):65–88, 2015.

[ABH+10] Jean-Raymond Abrial, Michael Butler, Stefan Hallerstede, Thai Son Hoang, Farhad Mehta, and Laurent Voisin. Rodin: an open toolset for modelling and reasoning in Event-B. *International journal on software tools for technology transfer*, 12(6):447–466, 2010.

[Abr05] Jean-Raymond Abrial. *The B-Book: Assigning programs to meanings*. Cambridge University Press, 2005.

[Abr10] Jean-Raymond Abrial. *Modeling in Event-B: system and software engineering*. Cambridge University Press, 2010.

[ACD90] Rajeev Alur, Costas Courcoubetis, and David L. Dill. Model-checking for real-time systems. In *Annual Symposium on Logic in Computer Science, LICS, 1990, Proceedings*, pages 414–425. IEEE Computer Society, 1990.

[ACM03] Jean-Raymond Abrial, Dominique Cansell, and Dominique Méry. A mechanically proved and incremental development of ieee 1394 tree identify protocol. *Formal aspects of computing*, 14(3):215–227, 2003.

[AD16] Étienne André and Benoît Delahaye. Consistency in parametric interval probabilistic timed automata. In *International Symposium on Temporal Representation and Reasoning, TIME 2016, Proceedings*, pages 110–119. IEEE Computer Society, 2016.

[ADF20] Étienne André, Benoît Delahaye, and Paulin Fournier. Consistency in parametric interval probabilistic timed automata. *J. Log. Algebr. Meth. Program.*, 110, 2020.

[ADL17] Mohamed Amine Aouadhi, Benoît Delahaye, and Arnaud Lanoix. Moving from Event-B to probabilistic Event-B. In *Annual ACM Symposium on Applied Computing, 2017, Proceedings*. ACM, 2017.

[ADL19a] Dimitri Antakly, Benoît Delahaye, and Philippe Leray. Graphical event model learning and verification for security assessment. In *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems, IEA/AIE 2019, Proceedings*, volume 11606 of *LNCS*, pages 245–252. Springer, 2019.

[ADL19b] Mohamed Amine Aouadhi, Benoît Delahaye, and Arnaud Lanoix. Introducing probabilistic reasoning within Event-B. *Software and Systems Modeling*, 18(3):1953–1984, 2019.

[AÉT+15] Olivier Aumont, Christian Éthé, Alessandro Tagliabue, Laurent Bopp, and Marion Gehlen. Pisces-v2: An ocean biogeochemical model for carbon and ecosystem studies. *Geoscientific Model Development Discussions*, 8(2), 2015.

[AGHH00] Ken Arnold, James Gosling, David Holmes, and David Holmes. *The Java programming language*, volume 2. Addison-Wesley Reading, 2000.

[AHV93] Rajeev Alur, Thomas A. Henzinger, and Moshe Y. Vardi. Parametric real-time reasoning. In *ACM Symposium on Theory of Computing, 1993, Proceedings*, pages 592–601, 1993.

[Ant20] Dimitri Antakly. *Apprentissage et Vérification Statistique pour la sécurité*. PhD thesis, Université de Nantes, France, 2020.

[APM09] Philippe Audebaud and Christine Paulin-Mohring. Proofs of randomized algorithms in coq. *Science of Computer Programming*, 74(8):568–589, 2009.

[Bac89] Ralph-JR Back. Refinement calculus, part II: Parallel and reactive programs. In *Workshop/School/Symposium of the REX Project (Research and Education in Concurrent Systems), 1989, Proceedings*, pages 67–93. Springer, 1989.

[BAD+19] Ran Bao, J. Christian Attiogbé, Benoît Delahaye, Paulin Fournier, and Didier Lime. Parametric statistical model checking of UAV flight plan. In *International Conference on Formal Techniques for Distributed Objects, Components, and Systems, FORTE 2019*, volume 11535 of *LNCS*, pages 57–74. Springer, 2019.

[BBB+10a] Ananda Basu, Saddek Bensalem, Marius Bozga, Benoît Caillaud, Benoît Delahaye, and Axel Legay. Statistical abstraction and model-checking of large heterogeneous

systems. In *International Conference on Formal Techniques for Distributed Systems, FORTE, 2010, Proceedings*, volume 6117 of *LNCS*, pages 32–46. Springer, 2010.

[BBB+10b] Ananda Basu, Saddek Bensalem, Marius Bozga, Benoît Delahaye, Axel Legay, and Emmanuel Sifakis. Verification of an afdx infrastructure using simulations and probabilities. In *International Conference on Runtime Verification, RV 2010, Proceedings*, volume 6418 of *LNCS*, pages 330–344. Springer, 2010.

[BBB+12] Ananda Basu, Saddek Bensalem, Marius Bozga, Benoît Delahaye, and Axel Legay. Statistical abstraction and model-checking of large heterogeneous systems. *International Journal on Software Tools for Technology Transfer*, 14(1):53–72, 2012.

[BBD+12] Saddek Bensalem, Marius Bozga, Benoît Delahaye, Cyrille Jégourel, Axel Legay, and Ayoub Nouri. Statistical model checking qos properties of systems with sbip. In *International Symposium on Leveraging Applications of Formal Methods, Verification and Validation, ISoLA, Proceedings*, volume 7609 of *LNCS*, pages 327–341. Springer, 2012.

[BBF+16] Pietro Belotti, Pierre Bonami, Matteo Fischetti, Andrea Lodi, Michele Monaci, Amaya Nogales-Gómez, and Domenico Salvagnin. On handling indicator constraints in mixed integer programming. *Computational Optimization and Applications*, pages 1–22, 2016.

[BC00] Didier Bert and Francis Cave. Construction of finite labelled transistion systems from B abstract systems. In *International Conference on Integrated Formal Methods, IFM 2000, Proceedings*, volume 1945 of *LNCS*, pages 235–254. Springer, 2000.

[BDA95] Andrea Bianco and Luca De Alfaro. Model checking of probabilistic and nondeterministic systems. In *International Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS, 1995, Proceedings*, pages 499–513. Springer, 1995.

[BDF+13] Nikola Benes, Benoît Delahaye, Uli Fahrenberg, Jan Kretínský, and Axel Legay. Hennessy-milner logic with greatest fixed points as a complete behavioural speci-

fication theory. In *International Conference on Concurrency Theory, CONCUR, 2013, Proceedings*, volume 8052 of *LNCS*, pages 76–90. Springer, 2013.

[BDF+18] Anicet Bart, Benoît Delahaye, Paulin Fournier, Didier Lime, Eric Monfroy, and Charlotte Truchet. Reachability in parametric interval markov chains using constraints. *Theor. Comput. Sci.*, 747:48–74, 2018.

[BDL+17] Anicet Bart, Benoît Delahaye, Didier Lime, Eric Monfroy, and Charlotte Truchet. Reachability in parametric interval Markov chains using constraints. In *International Conference on Quantitative Evaluation of Systems, 2017, Proceedings*, volume 10503 of *LNCS*, pages 173–189. Springer, 2017.

[BFG+14] Gilles Barthe, Cédric Fournet, Benjamin Grégoire, Pierre-Yves Strub, Nikhil Swamy, and Santiago Zanella Béguelin. Probabilistic relational verification for cryptographic implementations. In Suresh Jagannathan and Peter Sewell, editors, *Symposium on Principles of Programming Languages, POPL, 2014, Proceedings*, pages 193–206. ACM, 2014.

[BHP12] Benoît Barbot, Serge Haddad, and Claudine Picaronny. Coupling and importance sampling for statistical model checking. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS, 2012, Proceedings*, pages 331–346. Springer, 2012.

[bit] Bittorent description. `http://www.bittorrent.com/lang/fr/`.

[BK08] Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT Press, 2008.

[BLW13] Michael Benedikt, Rastislav Lenhardt, and James Worrell. LTL model checking of interval Markov chains. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS, 2013, Proceedings*, volume 7795 of *LNCS*, pages 32–46. Springer, 2013.

[BM13] Michael J. Butler and Issam Maamria. Practical theory extension in event-b. In *Theories of Programming and Formal Methods, 2013, Proceedings*, volume 8051 of *LNCS*, pages 67–81. Springer, 2013.

[BMS16] Luca Bortolussi, Dimitrios Milios, and Guido Sanguinetti. Smoothed model checking for uncertain continuous-time markov chains. *Information and Computation*, 247:235–253, 2016.

[BSST09] Clark W Barrett, Roberto Sebastiani, Sanjit A Seshia, and Cesare Tinelli. Satisfiability modulo theories. *Handbook of satisfiability*, 185:825–885, 2009.

[BvW89] Ralph JR Back and Joakim von Wright. Refinement calculus, part I: Sequential nondeterministic programs. In *Workshop/School/Symposium of the REX Project (Research and Education in Concurrent Systems), 1989, Proceedings*, pages 42–66. Springer, 1989.

[Can83] Georg Cantor. Überunendliche, lineare punktmannig faltigkeiten v [on infinite, linear point-manifolds (sets)]. *Math. Ann*, 21:545–591, 1883.

[CDL+10] Benoît Caillaud, Benoît Delahaye, Kim G. Larsen, Axel Legay, Mikkel L. Pedersen, and Andrzej Wasowski. Compositional design methodology with constraint markov chains. In *International Conference on the Quantitative Evaluation of Systems, QEST, 2010, Proceedings*, pages 123–132. IEEE Computer Society, 2010.

[CDL+11] Benoît Caillaud, Benoît Delahaye, Kim G. Larsen, Axel Legay, Mikkel L. Pedersen, and Andrzej Wasowski. Constraint markov chains. *Theor. Comput. Sci.*, 412(34):4373–4404, 2011.

[CHM97] David Maxwell Chickering, David Heckerman, and Christopher Meek. A bayesian approach to learning bayesian networks with local structure. In *International Conference on Uncertainty in artificial intelligence, 1997, Proceedings*, pages 80–89. Morgan Kaufmann Publishers Inc., 1997.

[CJGK+18] Edmund M Clarke Jr, Orna Grumberg, Daniel Kroening, Doron Peled, and Helmut Veith. *Model checking*. MIT press, 2018.

[CK15] Souymodip Chakraborty and Joost-Pieter Katoen. Model checking of open interval Markov chains. In *International Conference on Analytical and Stochastic Modelling Techniques and Applications, 2015, Proceedings*, volume 9081 of *LNCS*, pages 30–42. Springer, 2015.

[CLRS01] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. Introduction to algorithms second edition. *The Knuth-Morris-Pratt Algorithm*, 2001.

[cpl10] IBM ILOG CPLEX Optimizer, Last 2010.

[CS88] Wesley W Chu and Chi-Man Sit. Estimating task response time with contentions for real-time distributed systems. In *Real-Time Systems Symposium, 1988, Proceedings.*, pages 272–281. IEEE Computer Society, 1988.

[CSH08] Krishnendu Chatterjee, Koushik Sen, and Thomas A. Henzinger. Model-checking omega-regular properties of interval Markov chains. In *International Conference on Foundations of Software Science and Computational Structures, 2008, Proceedings*, volume 4962 of *LNCS*, pages 302–317. Springer, 2008.

[DEB17] Benoît Delahaye, Damien Eveillard, and Nicholas Bouskill. On the power of uncertainties in microbial system modeling: No need to hide them anymore. *MSystems*, 2(6):e00169–17, 2017.

[Del10] Benoît Delahaye. *Modular Specification and Compositional Analysis of Stochastic Systems. (Spécification Modulaire et Analyse Compositionnelle de Systèmes Stochastiques)*. PhD thesis, Université de Rennes 1, France, 2010.

[Del15] Benoît Delahaye. Consistency for parametric interval Markov chains. In *International Workshop on Synthesis of Complex Parameters, 2015, Proceedings*, volume 44 of *OASICS*, pages 17–32. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015.

[DFH+12] Benoît Delahaye, Uli Fahrenberg, Thomas A. Henzinger, Axel Legay, and Dejan Nickovic. Synchronous interface theories and time triggered scheduling. In *International Conference on Formal Techniques for Distributed Systems, FORTE 2012, Proceedings*, volume 7273 of *LNCS*, pages 203–218. Springer, 2012.

[DFL19] Benoît Delahaye, Paulin Fournier, and Didier Lime. Statistical model checking for parameterized models. working paper or preprint, February 2019.

[DFLL13a] Benoît Delahaye, Uli Fahrenberg, Kim Guldstrand Larsen, and Axel Legay. Refinement and difference for probabilistic automata. In *International Conference*

on Quantitative Evaluation of Systems, QEST, 2013, Proceedings, volume 8054 of *LNCS*, pages 22–38. Springer, 2013.

[DFLL13b] Benoît Delahaye, José Luiz Fiadeiro, Axel Legay, and Antónia Lopes. A timed component algebra for services. In *International Conference on Formal Techniques for Distributed Systems, FORTE, 2013, Proceedings*, volume 7892 of *LNCS*, pages 242–257. Springer, 2013.

[DFLL14a] Benoît Delahaye, Uli Fahrenberg, Kim G. Larsen, and Axel Legay. Refinement and difference for probabilistic automata. *Logical Methods in Computer Science*, 10(3), 2014.

[DFLL14b] Benoît Delahaye, José Luiz Fiadeiro, Axel Legay, and Antónia Lopes. Heterogeneous timed machines. In *International Colloquium on Theoretical Aspects of Computing, ICTAC, 2014, Proceedings*, volume 8687 of *LNCS*, pages 115–132. Springer, 2014.

[DGVJ12] Christian Dehnert, Daniel Gebler, Michele Volpato, and David N. Jansen. On abstraction of probabilistic systems. In *International Autumn School on Rigorous Dependability Analysis Using Model Checking Techniques for Stochastic Systems, ROCKS, 2012, Advanced Lectures*, volume 8453 of *LNCS*, pages 87–116. Springer, 2012.

[DJJ+15] Christian Dehnert, Sebastian Junges, Nils Jansen, Florian Corzilius, Matthias Volk, Harold Bruintjes, Joost-Pieter Katoen, and Erika Ábrahám. PROPhESY: A probabilistic parameter synthesis tool. In *International Conference on Computer Aided Verification, CAV, 2015, Proceedings*, volume 9207 of *LNCS*, pages 214–231. Springer, 2015.

[DJJL01] P. D'Argenio, B. Jeannet, H. Jensen, and K. Larsen. Reachability analysis of probabilistic systems by successive refinements. In *Joint International Workshop on Process Algebra and Probabilistic Methods, Performance Modeling and Verification, 2001, Proceedings*, volume 2165 of *LNCS*, pages 39–56. Springer, 2001.

[DJKV17] Christian Dehnert, Sebastian Junges, Joost-Pieter Katoen, and Matthias Volk. A storm is coming: A modern probabilistic model checker. In *International*

Conference on Computer Aided Verification, CAV, 2017, Proceedings, pages 592–600. Springer, 2017.

[DKL+11a] Benoît Delahaye, Joost-Pieter Katoen, Kim G. Larsen, Axel Legay, Mikkel L. Pedersen, Falak Sher, and Andrzej Wasowski. Abstract probabilistic automata. In *International Conference on Verification, Model Checking, and Abstract Interpretation, VMCAI, 2011, Proceedings*, volume 6538 of *LNCS*, pages 324–339. Springer, 2011.

[DKL+11b] Benoît Delahaye, Joost-Pieter Katoen, Kim G. Larsen, Axel Legay, Mikkel L. Pedersen, Falak Sher, and Andrzej Wasowski. New results on abstract probabilistic automata. In *International Conference on Application of Concurrency to System Design, ACSD, 2011, Proceedings*, pages 118–127. IEEE Computer Society, 2011.

[DKL+13] Benoît Delahaye, Joost-Pieter Katoen, Kim G. Larsen, Axel Legay, Mikkel L. Pedersen, Falak Sher, and Andrzej Wasowski. Abstract probabilistic automata. *Inf. Comput.*, 232:66–116, 2013.

[DLL+11a] Benoît Delahaye, Kim G. Larsen, Axel Legay, Mikkel L. Pedersen, and Andrzej Wasowski. Apac: A tool for reasoning about abstract probabilistic automata. In *International Conference on Quantitative Evaluation of Systems, QEST 2011, Proceedings*, pages 151–152. IEEE Computer Society, 2011.

[DLL+11b] Benoît Delahaye, Kim G. Larsen, Axel Legay, Mikkel L. Pedersen, and Andrzej Wasowski. Decision problems for interval markov chains. In *International Conference on Language and Automata Theory and Applications, LATA, 2011, Proceedings*, volume 6638 of *LNCS*, pages 274–285. Springer, 2011.

[DLL+12a] Benoît Delahaye, Kim G. Larsen, Axel Legay, Mikkel L. Pedersen, and Andrzej Wasowski. Consistency and refinement for interval Markov chains. *J. Log. Algebr. Program.*, 81(3):209–226, 2012.

[DLL+12b] Benoît Delahaye, Kim G. Larsen, Axel Legay, Mikkel L. Pedersen, and Andrzej Wasowski. New results for constraint markov chains. *Perform. Eval.*, 69(7-8):379–401, 2012.

[DLL13] Benoît Delahaye, Kim G. Larsen, and Axel Legay. Stuttering for abstract probabilistic automata. In *Logical Foundations of Computer Science*, volume 7734 of *LNCS*, pages 149–163. Springer, 2013.

[DLL14] Benoît Delahaye, Kim G. Larsen, and Axel Legay. Stuttering for abstract probabilistic automata. *J. Log. Algebr. Program.*, 83(1):1–19, 2014.

[DLP16] Benoît Delahaye, Didier Lime, and Laure Petrucci. Parameter synthesis for parametric interval Markov chains. In *International Conference on Verification, Model Checking, and Abstract Interpretation, 2016, Proceedings*, volume 9583 of *LNCS*, pages 372–390. Springer, 2016.

[DMB08] Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient smt solver. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems*, LNCS, pages 337–340. Springer, 2008.

[DREB98] Willem-Paul De Roever, Kai Engelhardt, and Karl-Heinz Buth. *Data refinement: model-oriented proof methods and their comparison*, volume 47. Cambridge University Press, 1998.

[DVJ07] Daryl J. Daley and David Vere-Jones. *An introduction to the theory of point processes: volume II: general theory and structure*. Springer Science & Business Media, 2007.

[EDLR16] Yrvann Emzivat, Benoît Delahaye, Didier Lime, and Olivier H. Roux. Probabilistic time petri nets. In *International Conference on Application and Theory of Petri Nets and Concurrency, PETRI NETS, 2016, Proceedings*, volume 9698 of *LNCS*, pages 261–280. Springer, 2016.

[Esp94] Javier Esparza. On the decidability of model checking for several μ-calculi and petri nets. In *Colloquium on Trees in Algebra and Programming, 1994, Proceedings*, pages 115–129. Springer, 1994.

[Fit12] Melvin Fitting. *First-order logic and automated theorem proving*. Springer Science & Business Media, 2012.

[FLDL18] José Luiz Fiadeiro, Antónia Lopes, Benoît Delahaye, and Axel Legay. Dynamic networks of heterogeneous timed machines. *Math. Struct. Comput. Sci.*, 28(6):800–855, 2018.

[GM16] Asela Gunawardana and Christopher Meek. Universal models of multivariate temporal point processes. In *International Conference on Artificial Intelligence and Statistics, 2016, Proceedings*, pages 556–563, 2016.

[GMX11] Asela Gunawardana, Christopher Meek, and Puyang Xu. A model for temporal dependencies in event streams. In *Advances in Neural Information Processing Systems, 2011, Proceedings*, pages 1962–1970, 2011.

[Gol98] Oded Goldreich. *Modern Cryptography, Probabilistic Proofs and Pseudorandomness*, volume 17 of *Algorithms and Combinatorics*. Springer, 1998.

[HA12] Hassan Haghighi and Mahsa Afshar. A z-based formalism to specify markov chains. *Computer Science and Engineering*, 2(3):24–31, 2012.

[Hai73] Frank A. Haight. *Handbook of the Poisson Distribution*. New York: Wiley, 1973.

[Ham50] Richard W Hamming. Error detecting and error correcting codes. *Bell System technical journal*, 29(2):147–160, 1950.

[HDR10] Dirk Husmeier, Richard Dybowski, and Stephen Roberts. *Probabilistic Modeling in Bioinformatics and Medical Informatics*. Springer, 2010.

[HH07] Stefan Hallerstede and Thai Son Hoang. Qualitative probabilistic modelling in Event-B. In *Integrated Formal Methods, 2007, Proceedings*, pages 293–312. Springer, 2007.

[HHS86] Jifeng He, C. A. R. Hoare, and Jeff W. Sanders. Data refinement refined. In *European Symposium on Programming, ESOP, 1986, Proceedings*, volume 213 of *LNCS*, pages 187–196. Springer, 1986.

[HHWZ10] Ernst Moritz Hahn, Holger Hermanns, Björn Wachter, and Lijun Zhang. PARAM: A model checker for parametric Markov models. In *International Conference on Computer Aided Verification, CAV, 2010, Proceedings*, volume 6174 of *LNCS*, pages 660–664. Springer, 2010.

[HMM05] Joe Hurd, Annabelle McIver, and Carroll Morgan. Probabilistic guarded commands mechanized in hol. *Electronic Notes in Theoretical Computer Science*, 112:95–111, 2005.

[HMU01] John E Hopcroft, Rajeev Motwani, and Jeffrey D Ullman. Introduction to automata theory, languages, and computation. *ACM Sigact News*, 32(1):60–65, 2001.

[HMZ+12] David Henriques, João Martins, Paolo Zuliani, André Platzer, and Edmund M. Clarke. Statistical model checking for markov decision processes. In *International Conference on Quantitative Evaluation of Systems, QEST, 2012, Proceedings*, pages 84–93. IEEE Computer Society, 2012.

[Hoa05] Thai Son Hoang. *The development of a probabilistic B-method and a supporting toolkit*. PhD thesis, The University of New South Wales, 2005.

[Hoa14] Thai Son Hoang. Reasoning about almost-certain convergence properties using Event-B. *Science of Computer Programming*, 81:108–121, 2014.

[Hur03] Joe Hurd. *Formal verification of probabilistic algorithms*. PhD thesis, University of Cambridge, Computer Laboratory, 2003.

[Hut20] M Hutson. Ai shortcuts speed up simulations by billions of times. *Science*, 367(6479):728, 2020.

[JL91] Bengt Jonsson and Kim Guldstrand Larsen. Specification and refinement of probabilistic processes. In *Symposium on Logic in Computer Science, LICS, 1991, Proceedings*, pages 266–277. IEEE Computer Society, 1991.

[JLS12] Cyrille Jegourel, Axel Legay, and Sean Sedwards. Cross-entropy optimisation of importance sampling parameters for statistical model checking. In *International Conference on Computer Aided Verification, CAV, 2012, Proceedings*, pages 327–342. Springer, 2012.

[KA98] Nagendra Kumar and Andreas G. Andreou. Heteroscedastic discriminant analysis and reduced rank hmms for improved speech recognition. *Speech communication*, 26(4):283–297, 1998.

[Kat07] Joost-Pieter Katoen. Abstraction of probabilistic systems. In *International Conference on Formal Modeling and Analysis of Timed Systems, FORMATS, 2007, Proceedings*, volume 4763 of *LNCS*, pages 1–3. Springer, 2007.

[KNP09] M. Kwiatkowska, G. Norman, and D. Parker. PRISM: Probabilistic Model Checking for Performance and Reliability Analysis. *ACM SIGMETRICS Performance Evaluation Review*, 36(4):40–45, 2009.

[KNP11] M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *International Conference on Computer Aided Verification, CAV, 2011, Proceedings*, volume 6806 of *LNCS*, pages 585–591. Springer, 2011.

[KNP12] M. Kwiatkowska, G. Norman, and D. Parker. Probabilistic verification of herman's self-stabilisation algorithm. *Formal Aspects of Computing*, 24(4):661–670, 2012.

[KR+88] Brian W Kernighan, Dennis M Ritchie, et al. *The C programming language*, volume 2. prentice-Hall Englewood Cliffs, NJ, 1988.

[LDB10] Axel Legay, Benoît Delahaye, and Saddek Bensalem. Statistical model checking: An overview. In *International Conference on Runtime Verification, RV, 2010, Proceedings*, volume 6418 of *LNCS*, pages 122–135. Springer, 2010.

[Leu06] Martin Leucker. Learning meets verification. In *International Symposium on Formal Methods for Components and Objects, FMCO, 2006, Proceedings*, pages 127–151. Springer, 2006.

[MHA05] Carroll Morgan, Thai Son Hoang, and Jean-Raymond Abrial. The challenge of probabilistic event b—extended abstract—. In *Formal Specification and Development in Z and B, ZB, 2005, Proceedings*, pages 162–171. Springer, 2005.

[MHA07] Richard Mayr, Noomene Ben Henda, and Parosh Aziz Abdulla. Decisive markov chains. *Logical Methods in Computer Science*, 3, 2007.

[Mil99] Robin Milner. *Communicating and mobile systems: the pi calculus*. Cambridge university press, 1999.

[MM06] Annabelle McIver and Charles Carroll Morgan. *Abstraction, refinement and proof for probabilistic systems*. Springer Science & Business Media, 2006.

[MR02] Kevin P. Murphy and Stuart Russell. Dynamic bayesian networks: representation, inference and learning. *University of California, Berkeley*, 2002.

[MR10] Rajeev Motwani and Prabhakar Raghavan. *Randomized algorithms*. Chapman & Hall/CRC, 2010.

[MSB11] Glenford J Myers, Corey Sandler, and Tom Badgett. *The art of software testing*. John Wiley & Sons, 2011.

[NBB+15] Ayoub Nouri, Saddek Bensalem, Marius Bozga, Benoît Delahaye, Cyrille Jégourel, and Axel Legay. Statistical model checking qos properties of systems with SBIP. *International Journal on Software Tools for Technology Transfer*, 17(2):171–185, 2015.

[Nei14] Daniel Neider. *Applications of automata learning in verification and synthesis*. PhD thesis, Hochschulbibliothek der Rheinisch-Westfälischen Technischen Hochschule Aachen, 2014.

[NPKS05] G. Norman, D. Parker, M. Kwiatkowska, and S. Shukla. Evaluating the reliability of NAND multiplexing with PRISM. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 24(10):1629–1637, 2005.

[NS06] G. Norman and V. Shmatikov. Analysis of probabilistic contract signing. *Journal of Computer Security*, 14(6):561–589, 2006.

[NSK02] Uri Nodelman, Christian R. Shelton, and Daphne Koller. Continuous time Bayesian networks. In *Proceedings of the Eighteenth conference on Uncertainty in artificial intelligence*, pages 378–387. Morgan Kaufmann Publishers Inc., 2002.

[PLSVS13] Alberto Puggelli, Wenchao Li, Alberto L. Sangiovanni-Vincentelli, and Sanjit A. Seshia. Polynomial-time verification of pctl properties of mdps with convex uncertainties. In *International Conference on Computer Aided Verification, CAV, 2013, Proceedings*, volume 8044 of *LNCS*, pages 527–542. Springer, 2013.

[pri] Prism modelchecker description. `http://www.prismmodelchecker.org/`.

[Put14] Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[RBW06] Francesca Rossi, Peter van Beek, and Toby Walsh. *Handbook of Constraint Programming (Foundations of Artificial Intelligence)*. Elsevier, 2006.

[REG+20] Simon Ramondenc, Damien Eveillard, Lionel Guidi, Fabien Lombard, and Benoît Delahaye. Probabilistic modeling to estimate jellyfish ecophysiological properties and size distributions. *Scientific Reports*, 10, 2020.

[RGH05] Shyamsundar Rajaram, Thore Graepel, and Ralf Herbrich. Poisson-networks: A model for structured point processes. In *International Workshop on artificial intelligence and statistics, 2005, Proceedings*, pages 277–284, 2005.

[RK16] Reuven Y Rubinstein and Dirk P Kroese. *Simulation and the Monte Carlo method*, volume 10. John Wiley & Sons, 2016.

[Ros09] Sheldon M. Ross. *A First Course In Probability*. Pearson, 2009.

[RR98] Michael K Reiter and Aviel D Rubin. Crowds: Anonymity for web transactions. *ACM transactions on information and system security (TISSEC)*, 1(1):66–92, 1998.

[RT13] Vinayak Rao and Yee Whye Teh. Fast MCMC sampling for Markov jump processes and extensions. *The Journal of Machine Learning Research*, 14(1):3295–3320, 2013.

[RV98] Didier Rémy and Jérôme Vouillon. Objective ml: An effective object-oriented extension to ml. *Theory and Practice of Object Systems*, 4(1):27–50, 1998.

[RVVDA08] Anne Rozinat, Manuela Veloso, and Wil M.P. Van Der Aalst. Using hidden markov models to evaluate the quality of discovered process models. *Extended Version. BPM Center Report BPM-08-10*, 161:178–182, 2008.

[S+78] Gideon Schwarz et al. Estimating the dimension of a model. *The annals of statistics*, 6(2):461–464, 1978.

[S+99] Michel F Sanner et al. Python: a programming language for software integration and development. *J. Mol. Graph Model*, 17(1):57–61, 1999.

[Shm04] V. Shmatikov. Probabilistic model checking of an anonymity system. *Journal of Computer Security*, 12(3/4):355–377, 2004.

[Spi88] J Michael Spivey. *Understanding Z: a specification language and its formal semantics*, volume 3. Cambridge University Press, 1988.

[ST96] Kaisa Sere and Elena Troubitsyna. Probabilities in action systems. In *Nordic Workshop on Programming Theory, NWPT, 1996, Proceedings*, pages 373–387, 1996.

[Sto02] Mariëlle Stoelinga. An introduction to probabilistic automata. *Bulletin of the EATCS*, 78(176-198):2, 2002.

[SVA04] Koushik Sen, Mahesh Viswanathan, and Gul Agha. Statistical model checking of black-box probabilistic systems. In *International Conference on Computer Aided Verification, CAV, 2004, Proceedings*, pages 202–215. Springer, 2004.

[SVA06] Koushik Sen, Mahesh Viswanathan, and Gul Agha. Model-checking Markov chains in the presence of uncertainties. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS, 2006, Proceedings*, volume 3920 of *LNCS*, pages 394–410. Springer, 2006.

[TBA06] Ioannis Tsamardinos, Laura E. Brown, and Constantin F. Aliferis. The max-min hill-climbing bayesian network structure learning algorithm. *Machine learning*, 65(1):31–78, 2006.

[TEF+05] Wilson Truccolo, Uri T Eden, Matthew R Fellows, John P Donoghue, and Emery N Brown. A point process framework for relating neural spiking activity to spiking history, neural ensemble, and extrinsic covariate effects. *Journal of neurophysiology*, 93(2):1074–1089, 2005.

[TRF03] Kishor S Trivedi, Srinivasan Ramani, and Ricardo Fricks. Recent advances in modeling response-time distributions in real-time systems. *Proceedings of the IEEE*, 91(7):1023–1037, 2003.

[TTL09] Anton Tarasyuk, Elena Troubitsyna, and Linas Laibinis. Reliability assessment in Event-B development. *NODES*, page 11, 2009.

[TTL10] Anton Tarasyuk, Elena Troubitsyna, and Linas Laibinis. Towards probabilistic modelling in Event-B. In *Integrated Formal Methods, IFM, 2010, Proceedings*, pages 275–289. Springer, 2010.

[TTL15] Anton Tarasyuk, Elena Troubitsyna, and Linas Laibinis. Integrating stochastic reasoning into Event-B development. *Formal Aspects of Computing*, 27(1):53–77, 2015.

[VDA14] W. Van Der Aalst. *Process Mining: Discovery, Conformance and Enhancement of Business Processes*. Springer, 2014.

[Vie15] Juan Pablo Vielma. Mixed integer linear programming formulation techniques. *SIAM Review*, 57(1):3–57, 2015.

[Vil92] Alain Villemeur. *Reliability, Availability, Maintainability and Safety Assessment, Assessment, Hardware, Software and Human Factors*, volume 2. Wiley, 1992.

[VM98] Christian Vogler and Dimitris Metaxas. Asl recognition based on a coupling between hmms and 3d motion analysis. In *International Conference on Computer Vision, 1998, Proceedings*, pages 363–369. IEEE Computer Society, 1998.

[Wal45] Abraham Wald. Sequential tests of statistical hypotheses. *The annals of mathematical statistics*, 16(2):117–186, 1945.

[WHA+16] Jacob Wurm, Khoa Hoang, Orlando Arias, Ahmad-Reza Sadeghi, and Yier Jin. Security analysis on consumer and industrial iot devices. In *Asia and South Pacific Design Automation Conference, ASP-DAC, 2016, Proceedings*, pages 519–524. IEEE Computer Society, 2016.

[WP13] Jeremy C Weiss and David Page. Forest-based point process for event prediction from electronic health records. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases, 2013, Proceedings*, pages 547–562. Springer, 2013.

[WT94] James A Whittaker and Michael G Thomason. A Markov chain model for statistical software testing. *IEEE Transactions on Software engineering*, 20(10):812–824, 1994.

[WTO+11] Tichakorn Wongpiromsarn, Ufuk Topcu, Necmiye Ozay, Huan Xu, and Richard M Murray. Tulip: A software toolbox for receding horizon temporal logic planning. In *ACM International Conference on Hybrid Systems: Computation and Control, 2011, Proceedings*, pages 313–314. ACM, 2011.

[Yil10] Emre Yilmaz. *Tool support for qualitative reasoning in Event-B*. PhD thesis, Master Thesis ETH Zürich, 2010, 2010.

[You05] Hakan L Younes. Verification and planning for stochastic processes with asynchronous events. Technical report, Carnegie-Mellon Univ Pittsburgh PA School of Computer Science, 2005.

APPENDIX A

RÉSUMÉ EN FRANÇAIS

Le but des sciences en général est d'étudier le fonctionnement de systèmes complexes afin de mieux les comprendre et, éventuellement, de pouvoir prédire leur comportement futur. Une part importante de cette étude est la réalisation d'un modèle, objet abstrait qui représente fidèlement les connaissances existantes à propos du système, et qui pourra alors être étudié et simulé à la place du système lui-même. Malheureusement, il est fréquent que les connaissances existantes à propos des systèmes soient incomplètes ou sujettes à incertitudes. Il est alors important d'inclure ces incertitudes à l'intérieur des modèles et de développer des techniques afin d'automatiser l'analyse de ces modèles. Ce document présente quatre contributions au domaine de la modélisation et de l'analyse de tels systèmes.

La première contribution, d'orientation plutôt théorique, présente une extension de la théorie des chaînes de Markov à intervalles dans le domaine paramétrique. Les chaînes de Markov à intervalles (IMC) sont un formalisme de modélisation pour les systèmes probabilistes discrets permettant de représenter une infinité de systèmes avec un objet abstrait fini. Dans une IMC, les probabilités de transition entre états sont abstraites par des intervalles à bornes constantes permettant de spécifier l'ensemble des probabilités pouvant être associées à ces transitions dans les systèmes concrets. Dans les chaînes de Markov paramétriques (pIMC), les bornes des intervalles ne sont plus nécessairement constantes mais peuvent prendre la forme de fonctions rationnelles d'un ensemble de paramètres. Dans ce cadre, nous étudions les différentes sémantiques pouvant être associées aux pIMC, et étudions en particulier le problème de la cohérence, *i.e.*, le fait qu'un pIMC donné représente bien un ensemble non vide de systèmes concrets. Nous prouvons dans un premier temps que ce problème est indépendant de la sémantique utilisée, puis proposons une méthode de résolution à base de contraintes. Finalement, nous considérons et solvons des problèmes plus complexes tels que l'accessibilité existentielle et universelle.

La deuxième contribution, d'orientation plutôt théorique elle aussi, concerne l'extension d'un formalisme de modélisation et de preuve de systèmes à évènements discrets : le B évènementiel. Jusqu'alors, le B évènementiel ne permettait d'introduire la notion de choix dans les systèmes qu'en utilisant le non-déterminisme. Notre contribution étend ce formalisme pour permettre de

prendre en compte, en plus du non-déterminisme, les choix probabilistes. Contrairement aux travaux existants, dans lesquels des choix probabilistes pouvaient uniquement être introduits au travers des affectations de variables, notre contribution permet d'introduire ces choix probabilistes à la place de n'importe quel choix non-déterministe présent dans le système (par exemple, en plus des affectations probabilistes, dans le choix entre les évènements ou encore dans la définition des paramètres). De plus, notre théorie autorise l'introduction de ces choix probabilistes à n'importe quelle étape du raffinement du système et s'intègre avec la théorie de preuves associée au B évènementiel. Nous introduisons ainsi la notion de raffinement probabiliste et proposons une étude de la notion de convergence presque certaine dans ce cadre.
La troisième contribution, d'orientation plus pratique, concerne la vérification de modèles exprimés dans le formalisme des chaînes de Markov à paramétrées (pMC). Pour cette contribution, nous partons de l'observation que la plupart des méthodes de vérification pour les pMC sont complexes et ne passent pas à l'échelle. C'est aussi le cas pour des formalismes de modélisation moins abstraits, tels que les chaînes de Markov (MC), mais il existe dans ce cas une technique statistique formelle permettant d'estimer la probabilité avec laquelle une MC satisfait une propriété donnée de manière très efficace. Cette technique, la vérification de modèle statistique, était jusqu'ici limitée aux modèles purement probabilistes et ne pouvait donc s'appliquer dans le cadre des pMC. Dans notre contribution, nous étendons la vérification de modèle statistique au cadre des pMC en s'inspirant de techniques statistiques telles que l'échantillonnage préférentiel. Nous présentons de plus un prototype d'outil, développé pour l'occasion, qui fait la preuve de concept que cette nouvelle technique est bien applicable en pratique et possède des propriétés intéressantes en termes de performances. Afin de montrer que notre technique passe à l'échelle, nous l'appliquons à un cas d'étude industriel : l'analyse du plan de vol d'un drone civil.
Finalement, la dernière contribution, d'orientation plutôt pratique elle aussi, s'attaque au problème de la modélisation automatisée. En effet le processus d'écriture d'un modèle est souvent fastidieux et requiert de l'expertise humaine. Dans cette contribution, notre but est le développement de techniques automatiques permettant de créer un modèle reproduisant les caractéristiques statistiques d'un ensemble de données. Nous présentons ainsi des techniques d'apprentissage reposant sur un formalisme de modélisation à temps continu : les "Recursive Timescale Graphical Event Models" (RTGEM). Dans ce contexte, nous proposons un algorithme permettant de générer automatiquement un modèle représentant le plus fidèlement possible les données en entrée tout en satisfaisant un ensemble de propriétés de sécurité fixées. La comparaison de ce modèle avec le modèle le plus représentatif des données permet ainsi d'évaluer, de manière automatique, le degré de sécurité associé aux données en entrée.

APPENDIX B

MAIN DEFINITIONS AND NOTATIONS

This appendix gathers the main notations and definitions introduced in the chapters of this document. It is aimed to be detached from the manuscript so the reader can use it while reading the main part of the document.
Due to the different topics treated in each chapter, remark that the definitions of some objects (Markov chains for instance) may slightly differ from one chapter to another.

Chapter 1: Parametric Interval Markov Chains

Variables, numbers and associated notions

- $X = \{x_1, \ldots, x_k\}$ is a set of variables and D_x, D_X are their associated domains/set of domains.
- A *rational function* f over X is a division of two (multivariate) polynomials g_1 and g_2 over X with rational coefficients, *i.e.*, $f = g_1/g_2$.
- $\mathbb{Q}$ is the set of rational numbers and $\mathbb{Q}_X$ is the set of rational functions over X.
- A *valuation* v over X is a set $v = \{(x,d) \mid x \in X, d \in D_x\}$ of elementary valuations. Valuations are extended to polynomials and rational functions.
- If S is a finite (or discrete) set, we write $\mathsf{Dist}(S)$ for the set of *probability distributions* over S, *i.e.*, the set of functions $\mu\colon S \to [0,1]$ such that $\sum_{s\in S}\mu(s) = 1$.
- $\mathbb{I}$ is the set containing all interval subsets of $[0,1]$.
- A is a set of atomic propositions (state labels), written in Greek alphabet.

Constraints

- An *atomic constraint* over X is a Boolean expression of the form $f(X) \bowtie g(X)$, with $\bowtie \in \{\leq, \geq, <, >, =\}$ and f and g two functions over variables in X. A *constraint* over X is a Boolean combination of atomic constraints over X.

Definition 1 (Constraint Satisfaction Problem - see p.27). *A* Constraint Satisfaction Problem *(*CSP*) is a tuple $\Omega = (X, D, C)$ where X is a finite set of variables, $D = D_X$ is the set of all domains associated to the variables from X, and C is a set of constraints over X.*

Markov chains and associated notions

Definition 2 (Markov Chain - see p.27). *A Discrete Time Markov Chain (*MC *for short) is a tuple $\mathcal{M} = (S, s_0, p, L)$, where S is a finite set of states containing the initial state s_0, $L\colon S \to 2^A$ is a labelling function, and $p\colon S \times S \to [0,1]$ is a probabilistic transition function such that for all $s \in S$, the function $p(s,.)$ is a distribution (*i.e.*, $s' \mapsto p(s,s') \in \mathsf{Dist}(S)$).*

- Given a state $s \in S$, $L(s)$ is called the *label* of s.
- `MC` is the set containing all discrete time Markov chains.
- A *run* of $\mathcal{M}$ is a sequence $w = s_1, s_2, \ldots$ with $p(s_i, s_{i+1}) > 0$, for all i.

Definition 3 (Probabilistic Bisimulation - see p.28). *Let $\mathcal{M} = (S, s_0, p, L)$ be a Markov chain. A* probabilistic *bisimulation on $\mathcal{M}$ is an equivalence relation $\mathcal{R}$ on S such that for all pair of states $(s_1, s_2) \in \mathcal{R}$, we have*

- *$L(s_1) = L(s_2)$, and*
- *$p(s_1, T) = p(s_2, T)$ for each equivalence class $T \in S/\mathcal{R}$.*

$\mathcal{M}_1$ and $\mathcal{M}_2$ are bisimilar *if and only if there exists a probabilistic bisimulation $\mathcal{R}$ on $\mathcal{M}_1 \cup \mathcal{M}_2$ with $(s_0^1, s_0^2) \in \mathcal{R}$.*

- Given a Markov chain $\mathcal{M}$, the probability of a *finite* sequence of states $w = s_0, s_1, \ldots, s_n$, written $\mathbb{P}_{\mathcal{M}}(w)$ is the product of the probabilities of the transitions transitions involved in this sequence, *i.e.*, $\mathbb{P}_{\mathcal{M}}(w) = p(s_0, s_1) \cdot p(s_1, s_2) \cdot \ldots \cdot p(s_{n-1}, s_n)$.
- $\mathsf{reach}_{s_0}(s) = \{ w \in S^* \mid w = s_0, \ldots s_n \text{ with } s_n = s \text{ and } s_i \neq s \ \forall 0 \leq i < n \}$ is the set of runs starting in s_0 and ending in s (without going through s before the end).
- $\mathbb{P}_{\mathcal{M}}(\Diamond s) = \sum_{w \in \mathsf{reach}_{s_0}(s)} \mathbb{P}_{\mathcal{M}}(w)$ is the overall probability of reaching state s in $\mathcal{M}$ (if $s \neq s_0$). This notation is extended to the probability of reaching a given state label or a given atomic proposition (see p.29).

Markov chain abstractions

Definition 4 (Abstraction Model - see p.29). *A Markov chain abstraction model (an abstraction model for short) is a pair $(\mathtt{L}, \models)$ where `L` is a nonempty set of models and $\models$ is a relation between `MC` and `L`. Let $\mathcal{P}$ be in `L` and $\mathcal{M}$ be in `MC`. We say that $\mathcal{M}$ implements $\mathcal{P}$ if and only if $(\mathcal{M}, \mathcal{P})$ belongs to $\models$ (*i.e.*, $\mathcal{M} \models \mathcal{P}$).*

Parametric Markov chains

Definition 5 (Parametric Markov Chain - see p.30). *A Parametric Markov Chain (*pMC *for short) is a tuple $I = (S, s_0, P, L, \mathbb{X})$ where S, s_0, and L are defined as for* MC*s, $\mathbb{X}$ is a set of variables (parameters) ranging over $[0,1]$, and $P\colon S \times S \to \mathbb{Q}_{\mathbb{X}}$ associates with each potential transition a parameterized probability.*

- pMC is the set containing all parametric Markov chains.
- The satisfaction relation $\models_{\mathtt{p}}$ between MC and pMC is defined by $\mathcal{M} \models_{\mathtt{p}} I$ if and only if $S = S'$, $s_0 = s_0'$, $L = L'$, and there exists a valuation v of $\mathbb{X}$ such that $p(s,s')$ equals $v(P(s,s'))$ for all s, s' in S.

Interval Markov chains

Definition 6 (Interval Markov Chain - see p.31). *An Interval Markov Chain (*IMC *for short) is a tuple $I = (S, s_0, P, L)$, where S, s_0, and L are defined as for* MC*s, and $P\colon S \times S \to \mathbb{I}$ associates with each potential transition an interval of probabilities.*

Definition 7 (Once-and-for-all Semantics - see p.31). *A* MC *$\mathcal{M} = (T, t_0, p, L^M)$ satisfies an* IMC *$I = (S, s_0, P, L^I)$ with the once-and-for-all semantics, written $\mathcal{M} \models^o_{\mathtt{I}} I$, if and only if $(T, t_0, L^M) = (S, s_0, L^I)$ and for all reachable states s in $\mathcal{M}$ and all state $s' \in S$, $p(s,s') \in P(s,s')$.*

Definition 8 (IMDP Semantics - see p.31). *A* MC *$\mathcal{M} = (T, t_0, p, L^M)$ satisfies an* IMC *$I = (S, s_0, P, L^I)$ with the IMDP semantics, written $\mathcal{M} \models^d_{\mathtt{I}} I$, if and only if there exists a mapping π from T to S such that*

(1) $\pi(t_0) = s_0$,

(2) $L^I(\pi(t)) = L^M(t)$ for all states $t \in T$, and

(3) $p(t,t') \in P(\pi(t), \pi(t'))$ for all pairs of states t, t' in T, where t is reachable in $\mathcal{M}$.

Definition 9 (At-every-step Semantics - see p.32). *A* MC *$\mathcal{M} = (T, t_0, p, L^M)$ satisfies an* IMC *$I = (S, s_0, P, L^I)$ with the at-every-step semantics, written $\mathcal{M} \models^a_{\mathtt{I}} I$ if and only if there exists a relation $\mathcal{R} \subseteq T \times S$ such that $(t_0, s_0) \in \mathcal{R}$, and whenever $(t,s) \in \mathcal{R}$, we have*

(1) the labels of s and t correspond: $L^M(t) = L^I(s)$,

(2) there exists a correspondence function $\delta_{(t,s)}\colon T \to (S \to [0,1])$ such that

(a) $\forall t' \in T$ if $p(t,t') > 0$ then $\delta_{(t,s)}(t')$ is a distribution on S

(b) For all $s' \in S$, $(\Sigma_{t' \in T}\, p(t,t') \cdot \delta_{(t,s)}(t')(s')) \in P(s,s')$, and

(c) For all $(t',s') \in T \times S$, if $\delta_{(t,s)}(t')(s') > 0$, then $(t',s') \in \mathcal{R}$.

Parametric interval Markov chains

- $\mathbb{X}$ is a finite set of parameters and v is a valuation over $\mathbb{X}$.
- $\mathbb{I}(\mathbb{Q}_{\mathbb{X}})$ is the set of all parametrized intervals over $[0,1]$.
- For all $J = [f_1, f_2] \in \mathbb{I}(\mathbb{Q}_{\mathbb{X}})$, $v(J)$ denotes the interval $[v(f_1), v(f_2)]$ if $0 \le v(f_1) \le v(f_2) \le 1$ and the empty set otherwise.

Definition 10 (Parametric Interval Markov Chain - see p.35). *A Parametric Interval Markov Chain (*pIMC *for short) is a tuple $\mathcal{P} = (S, s_0, P, L, \mathbb{X})$, where S, s_0, L, and $\mathbb{X}$ are defined as for* pMC*s, and $P\colon S \times S \to \mathbb{I}(\mathbb{Q}_{\mathbb{X}})$ associates with each potential transition a (parametric) interval.*

- $\texttt{pIMC}$ is the set containing all parametric interval Markov chains.
- Given a pIMC $\mathcal{P} = (S, s_0, P, L, \mathbb{X})$ and a valuation v, we write $v(\mathcal{P})$ for the IMC (S, s_0, P_v, L) obtained by replacing the transition function P from $\mathcal{P}$ with the function $P_v\colon S \times S \to \mathbb{I}$ defined by $P_v(s, s') = v(P(s, s'))$ for all $s, s' \in S$.
- The IMC $v(\mathcal{P})$ is called an *instance* of pIMC $\mathcal{P}$.
- $\models^a_{\texttt{pI}}$, $\models^d_{\texttt{pI}}$, and $\models^o_{\texttt{pI}}$ are three satisfaction relations for pIMCs.
  - $\mathcal{M} \models^a_{\texttt{pI}} \mathcal{P}$ iff there exists at least one IMC I instance of $\mathcal{P}$ such that $\mathcal{M} \models^a_{\texttt{I}} I$.
  - $\mathcal{M} \models^d_{\texttt{pI}} \mathcal{P}$ iff there exists at least one IMC I instance of $\mathcal{P}$ such that $\mathcal{M} \models^d_{\texttt{I}} I$.
  - $\mathcal{M} \models^o_{\texttt{pI}} \mathcal{P}$ iff there exists at least one IMC I instance of $\mathcal{P}$ such that $\mathcal{M} \models^o_{\texttt{I}} I$.
- The size of a pMC, IMC, or pIMC $\mathcal{L}$, written $|\mathcal{L}|$, corresponds to its number of states plus its number of transitions not reduced to 0, $[0,0]$ or $\varnothing$.
- $\texttt{Pred}(s) = \{\, s' \in S \mid P(s', s) \notin \{\, \varnothing, [0,0] \,\} \,\}$ are the predecessors of a given state s (extends to sets of states).
- $\texttt{Succ}(s) = \{\, s' \in S \mid P(s, s') \notin \{\, \varnothing, [0,0] \,\} \,\}$ are the successors of a given state s (extends to sets of states).

Model comparisons

Let $(\texttt{L}_1, \models_1)$ and $(\texttt{L}_2, \models_2)$ be two Markov chain abstraction models containing respectively $\mathcal{L}_1$ and $\mathcal{L}_2$.

- $\mathcal{L}_1$ is entailed by $\mathcal{L}_2$, written $\mathcal{L}_1 \sqsubseteq \mathcal{L}_2$, if and only if $\forall \mathcal{M} \models_1 \mathcal{L}_1, \exists \mathcal{M}' \models_2 \mathcal{L}_2$ such that $\mathcal{M}$ is bisimilar to $\mathcal{M}'$.
- $\mathcal{L}_1$ is (semantically) equivalent to $\mathcal{L}_2$, written $\mathcal{L}_1 \equiv \mathcal{L}_2$, if and only if $\mathcal{L}_1 \sqsubseteq \mathcal{L}_2$ and $\mathcal{L}_2 \sqsubseteq \mathcal{L}_1$.

Definition 11 (Succinctness - see p.36). *Let $(\texttt{L}_1, \models_1)$ and $(\texttt{L}_2, \models_2)$ be two Markov chain abstraction models. $\texttt{L}_1$ is at least as succinct as $\texttt{L}_2$, written $\texttt{L}_1 \leq \texttt{L}_2$, if and only if there exists a polynomial p such that for every $\mathcal{L}_2 \in \texttt{L}_2$, there exists $\mathcal{L}_1 \in \texttt{L}_1$ such that $\mathcal{L}_1 \equiv \mathcal{L}_2$ and $|\mathcal{L}_1| \leq p(|\mathcal{L}_2|)$. Moreover, $\texttt{L}_1$ is strictly more succinct than $\texttt{L}_2$, written $\texttt{L}_1 < \texttt{L}_2$, if and only if $\texttt{L}_1 \leq \texttt{L}_2$ and $\texttt{L}_2 \not\leq \texttt{L}_1$.*

Properties

Existential consistency

- A pIMC $\mathcal{P} = (S, s_0, P, L, \mathbb{X})$ is existentially consistent if and only if there exists a MC $\mathcal{M}$ satisfying $\mathcal{P}$.
- $\mathbf{C}_{\exists\mathbf{c}}(\mathcal{P})$ is a CSP encoding for verifying the existential consistency of pIMC $\mathcal{P}$, where the variables are as follows:
  - one variable π_p with domain $[0,1]$ per parameter p in $\mathbb{X}$,
  - one variable $\theta_s^{s'}$ with domain $[0,1]$ per transition (s,s') in $\{\{s\} \times \texttt{Succ}(s) \mid s \in S\}$, and
  - one Boolean variable ρ_s per state s in S.

Qualitative reachability

- A state label Γ is *existentially reachable* in pIMC $\mathcal{P} = (S, s_0, P, L, \mathbb{X})$ if and only if there exists an implementation $\mathcal{M}$ of $\mathcal{P}$ where Γ is reachable (*i.e.*, $\mathbb{P}_{\mathcal{M}}(\Diamond\Gamma) > 0$).
- Γ is *universally reachable* in $\mathcal{P}$ if and only if Γ is reachable in any implementation $\mathcal{M}$ of $\mathcal{P}$.
- $\mathbf{C}_{\exists\mathbf{r}}(\mathcal{P})$ is a CSP encoding, that extends $\mathbf{C}_{\exists\mathbf{c}}(\mathcal{P})$, for verifying these properties. In addition to the variables of $\mathbf{C}_{\exists\mathbf{c}}$, $\mathbf{C}_{\exists\mathbf{r}}$ contains:
  - one integer variable ω_s with domain $[0, |S|]$ per state s in S.

Quantitative reachability

- $\mathbf{C}'_{\exists\mathbf{r}}(\mathcal{P}, \Gamma)$ is a CSP that extends $\mathbf{C}_{\exists\mathbf{r}}(\mathcal{P}, \Gamma)$ with the folowing variables:
  - one Boolean variable λ_s per state s in S, and
  - one integer variable α_s with domain $[0, |S|]$ per state s in S.
- $\mathbf{C}_{\exists\tilde{\mathbf{r}}}(\mathcal{P}, \Gamma)$ is a CSP that extends $\mathbf{C}'_{\exists\mathbf{r}}(\mathcal{P}, \Gamma)$ with the following variables:
  - one variable γ_s with domain $[0,1]$ per state s in S.

Chapter 2: A Probabilistic Extension for Event-B

Transition Systems

- $\mathsf{Dist}(S)$ denotes the set of distributions over a given set S.
- A labelled transition system (LTS for short) is a tuple $\mathcal{M}$=(S, $Acts$, s_0, $\rightarrow$, AP, L) where S is a set of states, $s_0 \subseteq S$ is the initial state, $Acts$ is a set of actions, $\rightarrow \subseteq S \times Acts \times S$ is a transition relation AP is a set of atomic propositions, and $L : S \rightarrow 2^{AP}$ is a labelling function.
- A Probabilistic Labelled Transition System (PLTS for short) is a tuple $\mathcal{M}$=(S, s_0, $Acts$, P, AP, L) where S is a set of states, $s_0 \in S$ is the initial state, $Acts$ is a set of actions, AP is a set of atomic propositions, L: $S \rightarrow AP$ is a labelling function, and P: $S \times Acts \times S \rightarrow [0,1]$ is the transition probability function.
- A PLTS where, for each state $s \in S$ we have $\sum_{s' \in S, a \in Acts} P(s,a,s') = 1$ is a *Discrete Time Markov Chain*.

Event-B

- An Event-B model is a tuple M=($\bar{v}$,$\mathsf{I}(\bar{v})$,$\mathsf{V}(\bar{v})$,Evts, Init) where $\bar{v}$= $\{\mathsf{v}_1 \ldots \mathsf{v}_n\}$ is a set of variables, $\mathsf{I}(\bar{v})$ is an invariant, $\mathsf{V}(\bar{v})$ is an (optional) variant used for proving the convergence of the model, Evts is a set of events and $\mathsf{Init} \in \mathsf{Evts}$ is the initialisation event.
- An event is written e_i, with a set of parameters $\bar{t}$ = $\{\mathsf{t}_1 \ldots \mathsf{t}_n\}$ (optional), a guard $\mathsf{G}_i(\bar{t},\bar{v})$ and an action $\mathsf{S}_i(\bar{t},\bar{v})$.
- Assignment can be written $\mathsf{x} := \mathsf{E}(\bar{t}, \bar{v})$ (deterministic), $\mathsf{x} :| \mathsf{Q}(\bar{t},\bar{v},\mathsf{x},\mathsf{x}')$ (predicate), or $\mathsf{x} :\in \{\mathsf{E}_1(\bar{t},\bar{v}) \ldots \mathsf{E}_n(\bar{t},\bar{v})\}$ (non-deterministic).
- The action $\mathsf{S}_j(\bar{t},\bar{v})$ of a given event e_j may contain several assignments that are executed in parallel.

Probabilistic Event-B

- Probabilistic events are equipped with weights $\mathsf{W}_i(\bar{v})$.
- Predicate non-deterministic assignments are replaced with *predicate probabilistic assignments* written $\mathsf{x} :\oplus \mathsf{Q}_x(\bar{t},\bar{v},\mathsf{x}')$.
- Enumerated non-deterministic assignments are replaced with *enumerated probabilistic assignments* written $\mathsf{x} := \mathsf{E}_1(\bar{t},\bar{v})$ @ $\mathsf{p}_1 \oplus \ldots \oplus \mathsf{E}_m(\bar{t},\bar{v})$ @ p_m.

Definition 12 (Fully Probabilistic Event-B Model - see p.73). *A Fully probabilistic Event-B model is a tuple* M=($\bar{v}$,$\mathsf{I}(\bar{v})$, PEvts, Init) *where* $\bar{v}$= $\{\mathsf{v}_1 \ldots \mathsf{v}_n\}$ *is a set of variables,*

I($\bar{v}$) is the invariant, PEvts is a set of probabilistic events and Init is the initialisation event.

Semantics

Let M=($\bar{v}$, I($\bar{v}$), PEvts, Init) be a fully probabilistic Event-B model and σ be a valuation of its variables.

- Notation σ refers to variable valuations.
- Notation θ refers to parameter valuations when parameters are involved.
- Given a variable $x \in \bar{v}$, we write $[\sigma]x$ for the value of x in σ.
- Given an expression $E(\bar{v})$, we write $[\sigma]E(\bar{v})$ for the evaluation of $E(\bar{v})$ in the context of σ.
- Given an expression $E(\bar{t},\bar{v})$ over variables and parameters, we write $[\sigma,\theta]E(\bar{t},\bar{v})$ for the evaluation of $E(\bar{t},\bar{v})$ under parameter valuation θ and variable valuation σ.
- Given a probabilistic event e_i with a set of parameters $\bar{t}$ and a valuation σ of the variables, we write $T_\sigma^{e_i}$ for the set of parameter valuations θ such that the guard of e_i evaluates to true in the context of σ and θ.
- $P_{T_\sigma^{e_i}}$ is the uniform distribution on the set $T_\sigma^{e_i}$.
- $\mathcal{E}_{e_i}(x)$ is the set of all expressions that can be assigned to the variable x in the context of an enumerated probabilistic assignment $(x := E_1(\bar{t},\bar{v})@_{p_1} \oplus \ldots \oplus E_m(\bar{t},\bar{v})@_{p_m}$ $(m \geq 1))$.
- The probability of choosing an expression E_i in the above set is written $P_x^{e_i}(E_i) = p_i$.
- Given a probabilistic event e_i, we write $Var(e_i)$ for the set of variables in $\bar{v}$ that are modified by the action of e_i.
- If a variable x is modified by an enumerated probabilistic assignment of e_i, then we write $\mathcal{E}_{e_i}(x)|_{\sigma,\theta}^{\sigma'}$ for the set of expressions in $\mathcal{E}_{e_i}(x)$ such that their evaluation in the context of σ and θ returns the value of x in the valuation σ'.
- If a variable x is modified by a predicate probabilistic assignment $(x : \oplus Q_x(\bar{t},\bar{v},x'))$, then we write $\mathcal{V}_{\theta,\sigma}^{e_i}(x)$ for the set of values x' that make the predicate $Q_x(\bar{t},\bar{v},x')$ true when evaluated in σ and θ.
- $P_{\sigma,\theta}^{e_i}(x,\sigma')$ is the probability that a variable x is assigned the new value $[\sigma']x$ when executing probabilistic event e_i from the valuation σ and with parameter valuation θ.

Definition 13 (Fully Probabilistic Event-B operational Semantics - see p.78). *The operational semantics of a fully probabilistic Event-B model M=($\bar{v}$, I($\bar{v}$), PEvts, Init) is a PLTS $[\![M]\!] = (S, \mathsf{Acts}, P, s_0, AP, L)$ where $S, \mathsf{Acts}, s_0, AP$, and L are defined as in the standard LTS semantics of Event-B models (see Section 2.2.2), and $P : S \times \mathsf{Acts} \times S \to [0,1]$ is*

the transition probability function such that for a given state s, for all $e_i, s' \in \mathsf{Acts} \times S$*, we have* $P(s, e_i, s') = 0$ *if* $e_i \notin \mathsf{Acts}(s)$ *or* $\exists x \in X \backslash \{Var(e_i)\}$ *st* $[s]x \neq [s']x$ *and otherwise*

$$P(s, e_i, s') = \frac{[s]W_i(\vec{v})}{\sum_{e_j \in \mathsf{Acts}(s)} [s]W_j(\vec{v})} \times \sum_{\theta \in T_s^{e_i}} \left(P_{T_s^{e_i}}(\theta) \times \prod_{x \in Var(e_i)} P_{s,\theta}^{e_i}(x, s') \right)$$

Chapter 3: Statistical Model Checking for Parametric Systems

Background definitions

- The set of real numbers and the set of natural numbers are respectively written $\mathbb{R}$ and $\mathbb{N}$.
- Given two real numbers $a < b$, the closed, semi-open and open intervals representing all real values between a and b are respectively written $[a,b]$, $(a,b]$, $[a,b)$ and (a,b).
- A *Markov chain* is a tuple $\mathcal{M} = (S, s_0, P)$ where S is a denumerable set of states, $s_0 \in S$ is the initial state and $P : S \times S \to [0,1]$ is the transition probability function such that for all state $s \in S$, $\sum_{s' \in S} P(s,s') = 1$.
- A run of a Markov chain is a sequence of states $s_0, s_1 \ldots$ such that for all i, $P(s_i, s_{i+1}) > 0$.
- The length of a run, written $|\omega|$ represents the number of transitions it goes through (including repetitions).
- $\Gamma_{\mathcal{M}}(l)$ is the set of all finite runs of length l in $\mathcal{M}$, and $\Gamma_{\mathcal{M}}$ for the set of all finite runs in $\mathcal{M}$.
- The probability measure on the finite runs of $\mathcal{M}$ is written $\mathbb{P}_{\mathcal{M}}$.
- The probability of a finite run $\omega = s_0, s_1, \ldots s_n$ is $\mathbb{P}_{\mathcal{M}}(\omega) = \Pi_{i=1}^{n} P(s_{i-1}, s_i)$.
- Notation r is used for reward functions $r : \Gamma(l) \to \mathbb{R}$.
- We use the notation φ for properties on runs ($\varphi \subseteq \Gamma(l)$).
- Reachability properties are written $\mathbb{P}_{\mathcal{M}}(\Diamond^{\leq l} s)$.
- Safety properties are written Safety $\mathbb{P}_{\mathcal{M}}(\Box^{=l} E)$.
- Expected reward properties are written $\mathbb{E}^{l}_{\mathcal{M}}(r) = \sum_{\omega \in \Gamma(l)} \mathbb{P}_{\mathcal{M}}(\omega) r(\omega)$.

Parametric Markov chains

- $\mathbb{X}$ is a set of parameters (variables).
- $Poly(\mathbb{X})$ is the set of all real (multivariate) polynomials on $\mathbb{X}$.
- $v \in \mathbb{R}^{\mathbb{X}}$ stands for parameter valuations, and can be applied to polynomials by extension.
- A *Parametric Markov chain* is a tuple $\mathcal{M} = (S, s_0, P, \mathbb{X})$ such that S is a finite set of states, $s_0 \in S$ is the initial state, $\mathbb{X}$ is a finite set of parameters, and $P : S \times S \to Poly(\mathbb{X})$ is a parametric transition probability function.

- Given a pMC $\mathcal{M}$ and a parameter valuation $v \in \mathbb{R}^{\mathbb{X}}$, we write P_v for the transition probability function obtained under valuation v and $\mathcal{M}^v$ for the resulting structure.
- If $\mathcal{M}^v$ is a Markov chain (*i.e.*, if P_v satisfies the conditions of a Markov chain transition function), then we say that v is a *valid parameter valuation* w.r.t. $\mathcal{M}$.
- A run ω of $\mathcal{M}$ is a sequence of states $s_0, s_1, \ldots$ such that for all $i \geq 0$, $P(s_i, s_{i+1}) \neq 0$ (*i.e.*, the probability is either a strictly positive real constant or a function of the parameters).
- $\Gamma_{\mathcal{M}}(l)$ is the set of all finite runs of length l and $\Gamma_{\mathcal{M}}$ is the set of all finite runs of a pMC.

Monte Carlo analysis

- Given a pMC $\mathcal{M}$, we use the notation $r : \Gamma_{\mathcal{M}} \to \mathbb{R}$ for the reward function that we want to estimate using Monte Carlo.
- We use the notation f for the *normalization function* used in importance sampling : $f : S \times S \to [0,1]$. We says that f is *valid* w.r.t. pMC $\mathcal{M}$ if for all states s, $\sum_{s' \in S} f(s,s') = 1$.
- Given a pMC $\mathcal{M}$ and a valid normalization function f, we write $\mathcal{M}^f$ for the MC obtained from $\mathcal{M}$ by replacing P by f.
- $u_{\mathcal{M}}$ is the *uniform* normalization function, that yield uniform distributions on successor states following the original structure of $\mathcal{M}$.
- $Pa : \Gamma_{\mathcal{M}} \to Poly(\mathbb{X})$ is a parametric reward function defined inductively as follows: $Pa(s_0) = 1$ and for all run $\omega, s, s' \in \Gamma_{\mathcal{M}}$, $Pa(\omega, s, s') = Pa(\omega, s)P(s, s')$.
- r' is the parametric reward function that we use for computing the expectation of r and is defined as follows: Given any valid normalization function f and any run $\omega \in \Gamma_{\mathcal{M}}$, we define
$$r'(\omega) = \frac{Pa(\omega)}{\mathbb{P}_{\mathcal{M}^f}(\omega)} r(\omega).$$
- Y is a random variable, evaluated on runs of $\mathcal{M}^f$, such that $Y = r'(\omega) = \frac{Pa(\omega)}{\mathbb{P}_{\mathcal{M}^f}(\omega)} r(\omega)$.
- Given an experiment consisting of a set of runs $\{\omega_1, \ldots, \omega_n\}$, we write $\{Y_1, \ldots Y_n\}$ for the set of corresponding random variables.
- γ and σ^2 are the parametric functions giving, respectively, the mean value and variance of the variable Y_i.
- Given an unknown value/function such as γ, we write $\widehat{\gamma}$ for its *estimation* using our statistical methods.
- $\boldsymbol{\varphi}(z) = \int_{-\infty}^{z} exp(-x^2/2)dx/(\sqrt{2\pi})$ is the cumulative distribution function of the standard normal distribution $\mathcal{N}(0,1)$.

Chapter 4: Learning and Verification of Graphical Event Models

Statistical model checking

- We write S for a generic stochastic system.
- We use the notation φ for linear properties on the traces of a system.
- The random variable corresponding to the outcome of each simulation is written $\mathcal{B}_i$.
- The expected value of a given variable X is written $\mathbb{E}[X]$.
- The probability that this random variable is 1 (*i.e.*, that S satisfies the given property φ) is written $Pr[\mathcal{B}_i = 1] = p$.
- In qualitative SMC, the parameters are θ (threshold), α (Type-I error), β (Type-II error), and $[p_0; p_1]$, the indifference region. Usually, $p_0 = p - \delta$ and $p_1 = p + \delta$, and 2δ is the size of the symmetrical indifference region.
- In hypothesis testing, the hypothesis are $H_0 : p \leq \theta$ and $H_1 : p > \theta$.
- In quantitative SMC, the parameters are δ (precision) and ε (error rate).
- In all cases, the estimated value of an unknown variable p is written $\widehat{p}$.

Conditional intensity models

- We use the notation $t \in \mathbb{R}^+$ for timestamps and $l \in \mathcal{L}$ for labels.
- Given a totally ordered set S and a set of marks M, a *marked point process* (m.p.p) can be expressed as a sequence $\{(s_i, m_i) : i = 1, ..., n\}$ where for all i, $s_i \leq s_{i+1}$.
- An *event* is a pair (t, l) consisting of a timestamp and a label.
- An *event stream* is an m.p.p. where $S = \{t_i\}$ is a set of strictly increasing timestamps and $M = \{l_i\}$ is a set of event labels. It is written $x_{t^*} = (t_1, l_1), \ldots, (t_n, l_n)$, with $0 < t_i < t_{i+1} < t^*$ ($t^* = t_{n+1}$) for all $1 \leq i \leq n-1$.
- $t_0 = 0$ and $t^* = t_{n+1}$ are used as conventions for a stream consisting of n events $(t_i, l_i), 1 \leq i \leq n$.
- The length of an event stream is written $|x_{t^*}|$ and corresponds to the number of events (n in the above item).
- The history of the event streal at time t is the set of all the events that occurred before t: h_i denotes the ith history, with $h_i = h_{t_i} = (t_1, l_1), ..., (t_{i-1}, l_{i-1})$.
- When clear from the context, we will use x to denote data (event streams), h for a given history, and t for a givem timestamp.

Definition 14 (Conditional intensity function - see p.127). *Given an event stream x, a conditional intensity function* λ *is written as:*

$$\lambda(t \mid x) = \lim_{\Delta_t \to 0} \frac{\mathbb{E}[N(t, t+\Delta_t) | h_t]}{\Delta_t}$$

where $N(T)$ *denotes the number of points (events) occurring in a time interval* T *in x.*

Definition 15 (Conditional intensity model [DVJ07] - see p.128). *Formally, a conditional intensity model (CIM)* θ *is a set of indexed conditional intensity functions* $\{\lambda_l(t \mid x)\}_{l \in \mathcal{L}}$*. The data likelihood function is the joint density function of all the points in the m.p.p. that can be factorized into a product of all conditional intensity functions and can be written as:*

$$p(x \mid \theta) = \prod_{l \in \mathcal{L}} \prod_{i=1}^{n} \lambda_l(t_i \mid h_i; \theta)^{1_l(l_i)} e^{-\Lambda_l(t_i | h_i; \theta)}$$

where $\Lambda_l(t \mid h; \theta) = \int_{-\infty}^{t} \lambda_l(\tau \mid x; \theta) d\tau$ *for the data x and the indicator function* $1_l(l')$ *is one if* $l' = l$ *and zero otherwise.*

- In *Piecewise-Constant Conditional Intensity Models (PCIMs)*, intensity functions are constant in given time intervals (finite number of states that depend in the time t and the observed data x).
- For each label $l \in \mathcal{L}$, we consider a set of discrete states Σ_l.
- For each label l and each state $s \in \Sigma_l$ we consider a parameter λ_{ls}.
- We consider a mapping $\sigma_l : T \times X \longrightarrow \Sigma_l$ (with T the set of all possible times and X the set of all possible data) that determines the active state as a function of the chosen label and available data.

Definition 16 (Piecewise-constant conditional intensity models - see p.129). *A PCIM is defined by local structures* $S_l = (\Sigma_l, \sigma_l(t,x))$ *and local piecewise constant parameters* λ_{ls}*. Let* $S = \{S_l\}_{l \in \mathcal{L}}$ *be the set of all discrete states and* $\theta = \{\lambda_{ls}\}_{l \in \mathcal{L}, s \in \Sigma_l}$ *be the set of associated parameters. The PCIM data likelihood knowing the structure and the parameters in this case can be written as:*

$$p(x \mid S, \theta) = \prod_{l \in \mathcal{L}} \prod_{s \in \Sigma_l} \lambda_{ls}^{M_{ls}(x)} e^{-\lambda_{ls} T_{ls}(x)}$$

where $M_{ls}(x)$ *is the number of occurrence of events of type l while s is active in the event sequence x, and* $T_{ls}(x)$ *is the total duration while s was active for event type l.*

Graphical event models

Definition 17 (Graphical event model - see p.129). *A Graphical Event Model (GEM) is a tupple $\mathcal{G} = ((\mathcal{L}, E), \theta)$ where $(\mathcal{L}, E)$ is a directed graph and θ a set of parameters. The likelihood of the data g knowing the graph and its parameters is written as:*

$$p(x_{t^*} \mid t^*) = \prod_{i=1}^{|x_{t^*}|} \lambda_{l_i}(t_i \mid h_i) \prod_{i=1}^{|x_{t^*}|+1} e^{-\sum_{l \in \mathcal{L}} \int_{t_{i-1}}^{t_i} \lambda_l(\tau | h_i) d\tau},$$

with $h_{|x_{t^*}|+1}$ the entire history of the event stream, including x_{t^*}.

- The conditional intensity functions $\lambda_l(t \mid h)$ in GEMs are not *piecewise-constant*.
- The Markov property in the context of m.p.p.s. implies the following property on conditional intensity functions: For all l, h and t, $\lambda_l(t \mid h) = \lambda_l(t \mid [h]_{Pa(l)})$, where $Pa(l)$ are the parents of l in $\mathcal{G}$.

- A timescale is a set T of half-open intervals $(a, b]$ (with $a \geq 0$ and $b > a$) that form a partition of some interval $(c, t_h]$ (with $c \geq 0$), where t_h is the highest value of T.
- When an edge e is equiped with a timescale T, t_h is called the *horizon* of e.

Definition 18 (Timescale graphical event model - see p.131). *A TGEM $M = (\mathcal{G}, \mathcal{T})$ consists of a GEM $\mathcal{G} = ((\mathcal{L}, E), \theta)$ and a set of timescales $\mathcal{T} = T_{e_{(e \in E)}}$ corresponding to the edges E of the graph of $\mathcal{G}$.*

- $c_l(h, t)$ is the *parent count vector* of bounded counts (of occurrences) over the intervals in the timescales of the parents of l at time t.
- C_l is the set of all possible parent count vectors of label l.
- A given parent count vector yields a unique conditional intensity function for the corresponding parameter. The global conditional intensity functions are therefore piecewise-constant:
$$\lambda_l(t \mid h) = \{\lambda_{l, c_l(h,t)}\}_{c_l(h,t) \in C_l}.$$
- In the case of TGEMs, the likelihood of the data is written as follows:
$$p(x_{t^*} \mid t^*) = \prod_{l \in \mathcal{L}} \prod_{j \in C_l} \lambda_{l,j}^{n_{t^*,l,j}(x_{t^*})} e^{-\lambda_{l,j} d_{t^* l,j}(x_{t^*})},$$
where the sufficient statistics $n_{t^*,l,j}(x_{t^*})$ and $d_{t^* l,j}(x_{t^*})$ are the count of l-events and the durations, respectively, when the parent count vector was equal to j (a certain combination of parents).

- The sufficient statistics are formally written as:

$$n_{t^*,l,j}(x_{t^*}) = \sum_{i=1}^{|x_{t^*}|} \mathbf{1}(l_i = l)\mathbf{1}(c_l(h_i,t_i) = j)$$
$$d_{t^*,l,j}(x_{t^*}) = \sum_{i=1}^{|x_{t^*}|+1} \int_{t_{i-1}}^{t_i} \mathbf{1}(c_l(h_i,\tau) = j)d\tau$$

  where $t_{|x_{t^*}|+1}$ represents the duration d between the last event occurence and the final time t^* in the data.

Definition 19 (Recursive timescale graphical event model - see p.133). *The family of RTGEMs is defined recursively to be the finite closure of the empty model* $\mathcal{M}_0 = ((\mathcal{L},\{\}),\{\})$ *under a set of allowed operators* $O_F = \{add, split, extend\}$.

- The "add" operator adds a non-existing edge to a model and its corresponding timescale $T = (0,c]$, with c a constant (also called horizon).
- The "split" operator splits one interval $(a,b]$ in the timescale of a chosen edge into two intervals $(a, \frac{a+b}{2}], (\frac{a+b}{2}, b]$.
- The extend operator extends the horizon of a chosen edge by adding the interval $(t_h, 2t_h]$, with t_h being the previous horizon.
- The symmetric backward operators are $O_F^{-1} = \{reverse\_add, reverse\_split, reverse\_extend\}$.
- A finite consistent RTGEM learning procedure [GM16] is to do a forward greedy search (for model construction) followed by a backward greedy search (for model refinement), both based on model selection, and using data that is faithful.
- We choose to use the Bayesian Information Criterion (BIC) [S$^+$78], written as follows for a model M:

$$S_{t^*}(M) = log(p(x_{t^*} \mid t^*; M, \widehat{\lambda}_{t^*,l,j}(x_{t^*}))) - \sum_{l \in \mathcal{L}} \mid C_l \mid \cdot log(t^*)$$

  where $\widehat{\lambda}$ are the estimates of the λ parameters.
- Given a model M, the maximum likelihood estimate $\widehat{\lambda}$ of the λ parameters for M (knowing its structure) is as follows [GM16] :

$$\widehat{\lambda}_{t^*,l,j}(x_{t^*}) = \frac{n_{t^*,l,j}(x_{t^*})}{d_{t^*l,j}(x_{t^*})}.$$

Distance between RTGEM

- A timescale in an RTGEM can be represented by a vector $v = [0,a,b,c,\ldots]$ containing the successive endpoints values.

- Given two RTGEMs with the same set of labels $\mathcal{L}$, $G_1 = ((\mathcal{L}, E_1), \theta_1)$ and $G_2 = ((\mathcal{L}, E_2), \theta_2)$ and an edge $e \in E_1 \cap E_2$, we write v_1^e and v_2^e for the values of the respective timescales of e in G_1 and G_2.
- $v_{id}^e = |v_1^e \cap v_2^e|$ is the number of identical endpoints in the two vectors v_1^e and v_2^e.
- $v_{nid}^e = |v_1^e \setminus v_2^e| + |v_2^e \setminus v_1^e|$ is the number of endpoints that are not identical in the two vectors v_1^e and v_2^e.
- The elementary distance between the two respective timescales of the edge e is

$$d(v_1^e, v_2^e) = \frac{v_{nid}^e}{v_{nid}^e + v_{id}^e}$$

- The *Structural Hamming Distance (SHD)* between G_1 and G_2 is defined as follows:

$$\mathrm{SHD}(G_1, G_2) = |(E_1 \setminus E_2) \cup (E_2 \setminus E_1)| + \sum_{e \in E_1 \cap E_2} d(v_1^e, v_2^e)$$

- Consider two vectors v_1^e with size l and v_2^e with size k. The closest element from $v_{1_i}^e \in v_1^e$ in v_2^e is written $\mathbf{cl}(v_{1_i}^e, v_2^e) = argmin_{v_{2_p}^e}(|v_{1_i}^e - v_{2_p}^e|)$.
- The set of matches for a given edge $e \in E_1 \cap E_2$ is written V_{id}^e and defined as follows:

$$V_{id}^e = \{(v_{1_i}^e, v_{2_j}^e) \in v_1^e \times v_2^e : \mathbf{cl}(v_{1_i}^e, v_2^e) = v_{2_j}^e \wedge \mathbf{cl}(v_{2_j}^e, v_1^e) = v_{1_i}^e\}$$

- We write $V_{id,1}^e$ for the projection of V_{id}^e on v_1^e, *i.e.*, $V_{id,1}^e = \{v_{1,i}^e, \exists j, (v_{1,i}^e, v_{2,j}^e) \in V_{id}^e\}$.
- Similarly, we write $V_{id,2}^e$ for the projection of V_{id}^e on v_2^e.
- The set of unmatched endpoints is $V_{nid}^e = (v_1^e \setminus V_{id,1}^e) \cup (v_2^e \setminus V_{id,2}^e)$.
- We define $\mathbf{v}_{id}^e = |V_{id}^e|$ as the number of proximally matched endpoints and $\mathbf{v}_{nid}^e = |V_{nid}^e|$ as the number of unmatched enpoints.
- The proximal elementary distance between v_1^e and v_2^e is

$$d^*(v_1^e, v_2^e) = \frac{1}{\mathbf{v}_{nid}^e + \mathbf{v}_{id}^e} \left(\sum_{(v_{1_i}^e, v_{2_j}^e) \in V_{id}^e \setminus (0,0)} \frac{|v_{1_i}^e - v_{2_j}^e|}{min(v_{1_i}^e, v_{2_j}^e)} \right) + \frac{\mathbf{v}_{nid}^e}{\mathbf{v}_{nid}^e + \mathbf{v}_{id}^e}$$

- The *proximal distance measure* between G_1 and G_2 is defined as follows:

$$\mathrm{SHD}^*(G_1, G_2) = |(E_1 \setminus E_2) \cup (E_2 \setminus E_1)| + \sum_{e \in E_1 \cap E_2} d^*(v_1^e, v_2^e).$$

Experimentations

- The qualitative security property is of the form $\mathbb{P}(\varphi \mid M) > c$ where φ is a linear property (called a *query*) and c is a constant.

- Our algorithm starts with data $\mathcal{D}$, learns a first model M^o, then (if M^o does not satisfy the qualitative security property) a proximal model M^* satisfying the qualitative security property.
- The empty model is written $\perp$.

UNIVERSITE BRETAGNE LOIRE MATHSTIC

UNIVERSITÉ DE NANTES

Titre : Modélisation et Vérification de Systèmes Incertains

Mot clés : Modélisation, Vérification formelle, Systèmes probabilistes, Systèmes paramétrés

Résumé : Le but des sciences en général est d'étudier le fonctionnement de systèmes complexes afin de mieux les comprendre et, éventuellement, de pouvoir prédire leur comportement futur. Une part importante de cette étude est la réalisation d'un modèle, objet abstrait qui représente fidèlement les connaissances existantes à propos du système, et qui pourra alors être étudié et simulé à la place du système lui-même. Malheureusement, il est fréquent que les connaissances existantes à propos des systèmes soient incomplètes ou sujettes à incertitudes. Il est alors important d'inclure ces incertitudes à l'intérieur des modèles et de développer des techniques afin d'automatiser l'analyse de ces modèles. Ce document présente quatre contributions au domaine de la modélisation et de l'analyse de tels systèmes. Deux de ces contributions sont à vocation plutôt théorique et proposent des langages de modélisation permettant d'inclure les incertitudes à l'intérieur des modèles ainsi que des techniques de vérification associées. Les deux autres contributions sont plus pratiques, l'une proposant une technique de vérification statistique et prouvant son efficacité sur un cas d'étude industriel, et l'autre proposant une technique de construction automatique de modèle à partir d'un ensemble de données ainsi qu'une analyse de son niveau de sécurité.

Title: Modeling and Verification of Systems with Uncertainties

Keywords: Modeling, Formal verification, Probabilistic systems, Parametric Systems

Abstract: The goal of science in general is to study the functioning of complex systems in order to better understand them and potentially to be able to predict their future behaviour. An important part of this study is the realization of a model, an abstract object that faithfully represents existing knowledge about the system, and which can then be studied and simulated instead of the system itself. Unfortunately, existing knowledge about systems is often incomplete or subject to uncertainties. It is then important to include these uncertainties in the models and to develop techniques to automate the analysis of these models. This document presents four contributions to the field of modeling and analysis of such systems. Two of these contributions are somewhat theoreticaly-oriented, proposing modeling languages to include uncertainties in models and associated verification techniques. The other two contributions are more practical, one proposing a statistical verification technique and proving its efficiency on an industrial case study, and the other proposing a technique for automatied model construction from a data set and an analysis of its security level.