Model Checking Markov Chains: Techniques and Tools

Ivan S. Zapreev

Graduation committee:

Prof. Dr. C. Hoede (chairman)	University of Twente, The Netherlands
Prof. Dr. Ir. J.-P. Katoen (promotor)	RWTH Aachen / University of Twente, Germany / The Netherlands
Prof. Dr. E. Brinksma (promotor)	University of Twente / Embedded Systems Institute, The Netherlands
Prof. Dr. W. Fokkink	Vrije Universiteit Amsterdam, The Netherlands
Prof. Dr. Ir. B. R. Haverkort	University of Twente, The Netherlands
Prof. Dr. Ir. H. Hermanns	Saarland University, Germany
Prof. Dr. M. Kwiatkowska	Oxford University, England
Prof. Dr. J. C. van de Pol	University of Twente, The Netherlands
Dr. H. L. S. Younes	Google Incorporated, United States

IPA Dissertation Series 2008-11.
CTIT Ph.D.-Thesis Series No. 08-113, ISSN 1381-3617.
ISBN: 978-90-8570-298-6

The research reported in this dissertation has been carried out under the auspices of the Institute for Programming Research and Algorithmics (IPA) and within the context of the Center for Telematics and Information Technology (CTIT). The research funding was provided by the NWO Grant through the project: Model Checking Infinite-State Markov Chains (MC=MC).

Translation of the abstract: Ir. Tom Staijen and Dr. David N. Jansen.
Typeset by LaTeX.
Cover design: Airida Rekštytė.
Publisher: Wöhrmann Printing Service - http://www.wps.nl.

MODEL CHECKING MARKOV CHAINS: TECHNIQUES AND TOOLS

DISSERTATION

to obtain the doctor's degree
at the University of Twente, on the authority of
the rector magnificus, Prof. Dr. W.H.M. Zijm,
on account of the decision of the graduation committee
to be publicly defended
on Friday, March 7, 2008 at 15:00

by

Ivan S. Zapreev
born on 22 November 1979
in Novosibirsk, Russian Federation

The dissertation has been approved by the promotors:

Prof. Dr. Ir. Joost-Pieter Katoen and Prof. Dr. Ed Brinksma.

Abstract

Probabilistic model checking has been a successful research field in the recent decades. This dissertation deals with four important aspects of model checking Markov chains: the development of efficient model-checking tools, the improvement of model-checking algorithms, the efficiency of the state-space reduction techniques, and the development of simulation-based model-checking procedures.

We start by introducing MRMC, a model checker for discrete-time and continuous-time Markov reward models. It supports reward extensions of PCTL and CSL, and allows for the automated verification of properties concerning long-run and instantaneous rewards as well as cumulative rewards. In particular, it supports to check the reachability of a set of goal states (by only visiting legal states before) under a time and an accumulated reward constraint. Several numerical algorithms and extensions thereof are included in MRMC. We study the efficiency of the tool in comparison with several probabilistic model checkers by comparing verification times and peak memory usage for a set of standard case studies. The study considers the model checkers E $\vdash$MC^2, PRISM (sparse and hybrid mode), Ymer and VESTA, and focuses on fully probabilistic systems. Several of our experiments show significantly different run times and memory consumptions between the tools – up to various orders of magnitude – without, however, indicating a clearly dominating tool. For statistical model checking, Ymer prevails whereas for the numerical tools MRMC and PRISM (sparse) are rather close.

Further, we consider the time-bounded reachability problem for continuous-time Markov chains (CTMCs), the efficient algorithms for which are at the heart of probabilistic model checkers such as PRISM and E $\vdash$MC^2. For large time spans, on-the-fly steady-state detection is commonly applied. To obtain correct results (up to a given accuracy), it is essential to avoid detecting premature stationarity. We give a detailed account of criteria for steady-state detection in the setting of time-bounded reachability. This is done for forward- and backward-reachability algorithms. As a spin-off of this study, new results for on-the-fly steady-state detection during CTMC transient analysis are reported. Based on these results, a precise procedure for steady-state detection for time-bounded reachability is obtained. Experiments show the impact of these results in probabilistic model checking.

After that we study the effect of bisimulation minimization in model checking of monolithic discrete- and continuous-time Markov chains as well as variants thereof with rewards. Our results show that – as for traditional model checking – enormous state space reductions (up to logarithmic savings) may be obtained. While in traditional model checking, bisimulation minimisation pays off only rarely (because it is rather

slow), we find often enough that the verification time of the original Markov chain exceeds the minimisation time plus the verification time of the reduced chain. We consider probabilistic bisimulation as well as versions thereof that are tailored to the property to be checked.

We conclude our work by deriving new simulation-based techniques for model checking CSL properties on continuous-time Markov chains. The techniques provided so far were based on hypothesis testing and did not support model checking of all the main CSL operators. Our approach is based on discrete-event simulation and sequential confidence intervals. We provide model-checking algorithms for the main three CSL operators: time-interval until, unbounded until and steady-state. The experimental comparison of the suggested algorithms, integrated in MRMC, with the techniques based on hypothesis testing, implemented in Ymer and VESTA, shows that our approach is generally faster and that MRMC can handle more properties than the other statistical tools.

Samenvatting

Het onderzoeksgebied van probabilistisch *model checking* heeft de afgelopen decennia veel successen geboekt. Deze dissertatie behandelt vier belangrijke aspecten van model checking van Markovketens: de ontwikkeling van efficiënte model-checking-gereedschappen, de verbetering van model-checking-algoritmen, de efficiëntie van technieken om de toestandsruimte te verkleinen en de ontwikkeling van simulatiegebaseerde model-checking-methoden.

We beginnen met het introduceren van MRMC, een model-checker voor discrete-tijd en continue-tijd Markov-kostenmodellen. Hij ondersteunt kostenuitbreidingen van PCTL en CSL en kan automatisch eigenschappen betreffende lange-termijn en instantane kosten verifiëren, en ook betreffende cumulatieve kosten. Meer in bijzonder biedt hij de mogelijkheid om de bereikbaarheid van doeltoestanden te onderzoeken (via enkel toegestane toestanden) met een restrictie op tijdsduur en opgebouwde kosten. Meerdere numerieke algoritmen en uitbreidingen hiervan worden ondersteund door MRMC.

We vergelijken de efficiëntie van het gereedschap met verschillende probabilistische model checkers op basis van verificatieduur en maximaal geheugengebruik voor een verzameling standaardvoorbeelden. De studie kijkt naar E $\vdash$MC^2, PRISM (zowel sparse als hybride modus), Ymer en VESTA, en beperkt zich tot volledig probabilistische systemen. De experimenten tonen significante verschillen in tijdsduur en geheugengebruik tussen de gereedschappen – tot meerdere ordegroottes – zonder echter een duidelijk winnend gereedschap aan te kunnen wijzen. Bij statistisch model checking domineert Ymer, maar bij numerieke gereedschappen eindigen MRMC en PRISM (sparse) zeer dicht bij elkaar.

Vervolgens bespreken we het probleem van tijdsbeperkte bereikbaarheidseigenschappen; efficiënte algoritmen daarvoor vormen het hart van de probabilistische model checkers zoals PRISM en E $\vdash$MC^2. Bij lange tijdsbeperkingen wordt er vaak gebruik gemaakt van *on-the-fly* detectie van evenwichtstoestanden. Om correcte resultaten (met een bepaalde exactheid) te verkrijgen is het essentieel om voortijdige detectie te voorkomen. We geven een gedetailleerde lijst van criteria voor detectie van evenwicht in een context van tijdsbeperkte bereikbaarheid. Hierbij is gekeken naar zowel voorwaarts- als achterwaarts-werkende bereikbaarheidsalgoritmen. Als bijkomend resultaat van deze studie kunnen we nieuwe inzichten in on-the-fly evenwichtsdetectie bij transiente analyse van CTMCs vermelden. Met behulp van deze resultaten komen we tot een precieze procedure voor het detecteren van evenwichtstoestanden bij tijdsbeperkte bereikbaarheid. Experimenten laten de uitwerkingen van deze resultaten zien in probabilitisch model checking.

Daarna bestuderen we de effecten van bisimulatie-minimalisatie voor model checking van monolithische discrete- en continue-tijd Markovketens maar ook varianten daarvan met kosten. Onze resultaten laten zien dat – zoals ook voor traditionele model verificatie – de toestandsruimte sterk verkleind kan worden (tot logaritmische besparingen). Maar terwijl in traditioneel model checking bisimulatie-minimalisatie slechts zelden loont (omdat zijzelf relatief langzaam is), vinden we hier vaak de situatie dat de tijd voor verificatie van de oorspronkelijke Markovketen langer is dan de tijd voor minimalisatie plus de tijd voor verificatie van het gereduceerde model. We bespreken probabilitische bisimulatie en variaties daarop die zijn aangepast aan de te controleren eigenschap.

We ronden ons werk af met het afleiden van nieuwe simlatie-gebaseerde technieken voor model checking van CSL-eigenschappen en continue-tijd Markovketens. De tot dusver bestaande technieken waren gebaseerd op testen van hypothesen en ondersteunden niet alle belangrijke CSL-operatoren. Onze aanpak is gebaseerd op discrete-gebeurtenissen-simulatie en sequentiele confidentie-intervallen. We tonen model-checking-algoritmen voor de belangrijkste drie CSL-operatoren: tijds-interval until, niet tijdsbeperkte until en evenwichtstoestands-operator. In experimenten vergelijken wij de voorgestelde algoritmen, geïntegreerd in MRMC, met de technieken gebaseerd op testen van hypothesen, geïmplementeerd in Ymer en VESTA; daaruit blijkt dat onze aanpak over het algemeen sneller is en dat MRMC meer eigenschappen aankan dan de andere statistische gereedschappen.

Acknowledgments

I am thankful to many people for their help and support during my work on this dissertation. Below, I would like to acknowledge those who participated in my supervision, research, and everyday life.

First of all I would like to thank my direct supervisor Joost-Pieter Katoen for guiding me through the not always serene waters of research. Without his care, encouragement and steering this thesis would never be written. Next, I should mention colleagues who contributed to the presented work in many different ways, such as: joint papers, reviews of the thesis chapters, valuable discussions, finding serious flaws in early versions of my work, assistance with the MRMC tool development, and etc. In order to reflect everyone's input (chapter wise), I summarize it in the table below.

Name	Papers	Reviews	Discussions	Flaws	MRMC
Prof. Dr. Ed Brinksma		Ch. 1 – 7			
Prof. Dr. Ir. Boudewijn Haverkort			Ch. 6		
Dr. Ir. Pieter-Tjerk de Boer			Ch. 6	Ch. 6	
Dr. Henrik Bohnenkamp		Ch. 5, 6	Ch. 3, 5, 6		
Dr. David N. Jansen	Ch. 2, 4	Ch. 5, 6	Ch. 2 – 6	Ch. 3, 6	Ch. 2
Dr. Mariëlle Stoelinga	Ch. 2		Ch. 2		
Dr. Håkan L. S. Younes			Ch. 6, 7	Ch. 6	
MSc. Tim Kemna	Ch. 4				Ch. 4
MSc. Maneesh Khattri	Ch. 2		Ch. 3, 2		Ch. 2
MSc. Marcel Oldenkamp	Ch. 2		Ch. 2		
Christina Jansen					Ch. 2, 7

I want to acknowledge our secretary Joke Lammerink for all her care and help. She is the one who makes the clock of the Formal Methods and Tools group ticking. I am also grateful to Miranda van Wijk who is the most professional P&O-advisor I have ever seen.

I am sincerely grateful to my friends Henrik Bohnenkamp, Erika Ábrahám, Tom Staijen, Julius Schwartzenberg, Rajasekhar Kakumani and Tomas Krilavičius for all the wonderful time we spent together and all the support they have given me. Dear friends, you made the last four years of my life worth it. I could seldom meet my Russian friends Slava Klochkov and Sergey Brazhnik, but we have kept in touch over the years and therefore I thank them.

Last but not least, I would like to thank my family. My beautiful wife Galina, who is the wisest woman I have ever met. My beloved father and mother, who gave me life and made me who I am. My brother Peter, whom I am proud of. My grandmother Belousova Nina Viktorovna and grandfather Belousov Anatoly Fedorovich, who taught me to love mathematics and showed me the wonderful world of science. Thank you all for being in my heart.

Contents

Introduction

In our everyday life we become more and more confronted with information technology, either explicitly, when dealing with personal computers or mobile phones, or implicitly, when using TVs, cars, trains, etc. It goes without saying that now our lives are more than ever dependent on the reliability of various software and hardware components.

It is indeed just a small inconvenience if a mobile phone malfunctions or a video camera fails to respond accurately to its controls, but a mistake in software controlling a nuclear power plant or a radiation therapy machine can have dramatic consequences. Moreover, even when not a matter of life and death, errors in software and hardware can be financially serious if a faulty product has to be recalled or replaced. For example, small mistakes in Intel's Pentium floating-point division unit and in the flight control of Ariane-5 missile both caused losses worth of hundreds of millions of US dollars.

This is why *system validation*, the process of determining the correctness of system specifications, designs and implementations is of the utmost importance. It is well-known that complexity of developed systems grows rapidly. Nevertheless, current practices, for instance in software engineering, show that system designs are mostly validated by humans with very little use of tools and especially tools with a sound mathematical basis. All that facilitates the need in techniques and tools for an automated system validation.

Further, in Section 1 we briefly discuss various system-validation techniques along with possible levels of their automation. One of them, model checking, is considered in more detail in Section 2. There we specifically talk about model checking of Markov chains, as it is the main topic of this dissertation. Finally, in Section 3 we present a high-level outline of our research.

1 System validation

System validation techniques can be divided into four main categories: testing, simulation, formal verification, and model checking.

Testing is performed on a real implementation of the system or on its prototype. The technique is an operational way of checking the conformance between the system implementation and the abstract system specification. Therefore, only a partial evaluation of the system design is possible.

Simulation is similar to testing, but is based on an executable system model and thus only allows for a quick and shallow evaluation of the design quality. Clearly, this approach is not suitable for finding subtle system errors.

Formal verification mathematically proves the correctness of the design, provided in the form of the system model, with respect to a formal specification. In practice, writing a complete formal proof of correctness for real-world hardware and software is difficult. This problem is tackled by automatic and semi-automatic approaches to formal verification. Unfortunately, most of the suggested techniques require detailed human guidance.

Model checking is a technique that can be fully automated. In this approach desired system properties, stated in some logical formalism (such as temporal logic), are verified against the system model, e. g. employing an exhaustive state-space exploration.

It is clear that for being successful any approach to system validation must allow for a good degree of automation. Therefore, in the field of testing there are algorithms for test generation and test selection based on the system specification. In formal verification there are proof assistants, proof checkers and theorem provers that, however, often require quite some expertise from the user. Model checking is perhaps the only technique that provides full support for automatic verification. All model-checking algorithms, implemented in software, do not require any guidance from the user. This is why model checking raises an increasing interest in industry – various companies, e. g. Intel and IBM, have research groups working on this topic and develop their own in-house model checkers.

To put it in a nutshell, model checking is an automated technique that establishes whether certain qualitative properties such as deadlock-freedom or request-response requirements ("does a request always lead to a response?") hold in a model of the system under consideration. Such models are typically transition systems that specify how the system may evolve during execution. Properties are usually expressed in temporal extensions of propositional logic, such as Linear Time Logic (LTL) [114] or Computational Tree Logic (CTL) [32]. In the remainder of this dissertation we will concentrate on model checking of *probabilistic* systems.

2 Model checking Markov chains

Since the seminal work of Hansson and Jonsson [56], adapting model checking to probabilistic systems has been a rather active research field. This has resulted in efficient algorithms for model-checking discrete- and continuous-time Markov Chains (DTMCs and CTMCs), their reward (cost) extensions, as well as Markov decision processes.

The applicability of probabilistic model checking ranges from areas such as randomized distributed algorithms to planning and AI, security [109], and even biological process modeling [95]. Probabilistic model-checking engines have been integrated in existing tool chains for widely used formalisms such as stochastic Petri nets [38], Statemate [19], the stochastic process algebra PEPA [67], and a probabilistic variant of Promela [9]. Popular logics are Probabilistic CTL (PCTL) [56] and Continuous Stochastic Logic (CSL) [8].

The typical kind of properties that can be checked are time-bounded reachability properties – "Does the probability to reach a certain set of goal states (by avoiding bad states) within a maximal time span exceed 0.5?" – and long-run averages – "In equilibrium, does the likelihood to leak confidential information remain below 10^{-4}?" Extensions for reward-based models allow for checking more involved properties that

refer to e. g., the expected cumulated reward or the instantaneous reward rate of computations. Intricate combinations of numerical or simulation techniques for Markov chains, optimization algorithms, and traditional LTL or CTL model-checking algorithms result in simple, yet efficient verification procedures. Verifying time-bounded reachability properties on models of tens of millions of states usually is a matter of minutes or even seconds.

Unfortunately, like in the traditional setting, probabilistic model checking suffers from the state-space explosion problem: the number of states grows exponentially in the number of system components and cardinality of data domains. This poses three main directions in further development of probabilistic model checking: advances of efficient state-space reduction techniques, improvement of the model-checking algorithms' performance, and introduction of simulation-based verification procedures. This work tackles all these aspects including realization of verification algorithms in a new probabilistic model checker.

3 Outline of the dissertation

This dissertation is divided into four parts. Part I contains results related to numerical model checking of PCTL, CSL, their reward extensions, and tool development. Part II is devoted to new techniques in model checking CSL using discrete-event simulation. Part III concludes the main scope of the thesis by summarizing key results and outlining where, in our opinion, further research activities should be undertaken. Part IV contains supplementary material such as theorem proofs and tool-profiling data. Further, we describe the content of the main rubrics of this dissertation, i. e. Part I and II. At the end, we present a list of publications this work resulted in.

Part I: Numerical Model Checking.

We begin with Chapter 1 containing necessary preliminary material on model-checking Markov chains. In this chapter, we first introduce Markov chains along with the transient and stationary probabilities, and numerical methods for computing them. Then, we proceed with a brief introduction into model checking DTMCs, CTMCs, and reward extensions thereof. The rest of preliminaries is devoted to a description of case studies and various probabilistic model checkers used in this work for experiments and comparison.

In Chapter 2, we report on a new probabilistic model checker named Markov Reward Model Checker (MRMC). This tool is used as an experimental platform for evaluating our algorithms and comparing their efficiency with techniques implemented in other model checking tools. Chapter 2 contains information about the functionality of MRMC, its performance, implementation metrics and use in third party projects. We also provide a comparative experimental study of MRMC and a set of state-of-the-art probabilistic model checkers.

On-the-fly steady-state detection is an optimization technique used in model checking of time-bounded reachability properties on CTMCs [87]. For large time spans, on-the-fly steady-state detection is commonly applied but for obtaining correct results (up to a given accuracy), it is essential to avoid detecting premature stationarity. The

latter, however, is not always the case. Therefore, in Chapter 3 we give a detailed account of criteria for steady-state detection in the setting of time-bounded reachability considering both forward and backward reachability algorithms. In essence, we improve on-the-fly steady-state detection for CTMC transient analysis and time-bounded reachability problem by refining the error bounds and deriving a precise steady-state detection procedure.

It is a well known fact that in traditional model checking bisimulation minimization allows for enormous state-space reductions (up to exponential savings) but is impractical due to high minimization times. So far, the impact of bisimulation minimization on probabilistic model checking was left undisclosed. In Chapter 4, we study the effect of bisimulation minimization in model checking of DTMCs, CTMCs, and their reward extensions. In our work we consider probabilistic bisimulation as well as versions thereof that are tailored to the property to be checked.

Part II: Model Checking by Discrete Event Simulation.

Numerical analysis and statistical techniques based on sampling and Monte Carlo simulation are two distinct approaches to model checking Markov chains. Recent developments in model checking of CTMCs resulted in simulation-based algorithms for model checking a subset of CSL that, however, does not include all the main operators of this logic. The suggested algorithms employ simple and sequential hypothesis testing and do not suffer from the state-space explosion. Our contribution in this field is discussed in Part II.

In Chapter 5 we provide the preliminary material required for Chapters 6 and 7. In this chapter, we start with discussing *point estimates* and *confidence intervals* for mean values of random variables. Further, we consider their application in *terminating* and *steady-state* simulations, an approach of Hordijk et al. [70] for simulating CTMCs, and confidence intervals for Bernoulli trials.

Based on the techniques discussed in Chapter 5, we propose an approach to model checking CSL using discrete-event simulation and sequential confidence intervals. The new algorithms for model checking all the main operators of CSL are devised in Chapter 6. To show the feasibility of our approach we perform an experimental comparison of the suggested techniques, implemented in MRMC, and the ones based on hypothesis testing, implemented in statistical model-checking tools Ymer and VESTA. The results of this comparison are provided in Chapter 7.

Published results.

Most results of Part I have been published as:

- Joost-Pieter Katoen, Maneesh Khattri, and Ivan S. Zapreev. A Markov Reward Model Checker. In *Quantitative Evaluation of Systems (QEST)*, pages 243–244. IEEE Computer Society, 2005.
- Joost-Pieter Katoen and Ivan S. Zapreev. Safe On-The-Fly Steady-State Detection for Time-Bounded Reachability. In *Quantitative Evaluation of Systems (QEST)*, pages 301–310. IEEE Computer Society, 2006.

- Joost-Pieter Katoen, Tim Kemna, Ivan S. Zapreev, and David N. Jansen. Bisimulation Minimization Mostly Speeds Up Probabilistic Model Checking. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 4424 of LNCS, pages 87–101. Springer, 2007.
- David N. Jansen, Joost-Pieter Katoen, Marcel Oldenkamp, Marielle Stoelinga, and Ivan S. Zapreev. How Fast and Fat Is Your Probabilistic Model Checker? In *Haifa Verification Conference (HVC)*, volume 4899 of LNCS, pages 65–79. Springer, 2008.

The results presented in Part II are new and therefore have not yet been published.

Part I

Numerical Model Checking

Chapter 1

Preliminaries

In this chapter, we introduce the preliminary material used throughout the thesis. The reader is assumed to be familiar with classical probability theory as can be found in [42, 18]. We start with Section 1.1 introducing the discrete- and continuous-time finite-state Markov chains as they are the main underlying models in our research. In addition, we discuss the transient and stationary probabilities of Markov chains along with the ways of computing them. This material is immediately put to use in Section 1.2. There we explain the main concepts of model checking Markov chains and show how some of the model-checking procedures can be reduced to graph analysis and computing transient and stationary probabilities. We conclude Section 1.2 by briefly describing model checking of Markov reward models. Further, in Section 1.3 we discuss real-life systems that can be modeled as Markov chains. These systems are commonly used as benchmark problems in probabilistic model checking and therefore we only concentrate on a high-level description thereof. The set of most known tools that allow for probabilistic verification is presented in Section 1.4. These tools and models are used for comparative studies and experiments provided in this dissertation. Section 1.5 concludes.

1.1 Markov chains

Markov chains are a special case of stochastic processes. Therefore, we first briefly introduce the latter ones, and explain the set of conditions needed for a stochastic process to be called a Markov chain. Further we proceed with a classification of Markov chain states and give definitions of discrete- and continuous-time Markov chains along with the ways of computing their transient and stationary probabilities. For more information on Markov chains we refer to standard textbooks such as [134]. Most of the results provided in this section can be found in [59, 131]

A *stochastic process* is a collection of random variables $\{\mathcal{X}_t \mid t \in \mathcal{T}\}$ defined on a probability space and indexed by a parameter t which can take values in $\mathcal{T}$. Typically t is assumed to represent time. The values of $\mathcal{X}_t$ are called *states*. The set of all possible states of the stochastic process is called the *state space* and is denoted as S. Clearly, the state space can be either continuous or discrete. In the former case we deal with a *continuous-state stochastic process* and in the latter with a *discrete-*

state stochastic process, which is called a *chain* and for convenience we assume that $S = \{0,\ 1,\ 2,\ \dots\}$. A similar classification can be made regarding the index set $\mathcal{T}$. A denumerable set leads to the *discrete-time stochastic process* whereas a continuous set leads to the *continuous-time stochastic process*.

A stochastic process is called a *Markov process* if for any $t_0 < \ldots < t_n < t_{n+1}$ the distribution of $\mathcal{X}_{t_{n+1}}$, given the values of $\mathcal{X}_{t_0}, \ldots, \mathcal{X}_{t_n}$ ($s_0, \ldots, s_n \in S$ respectively), only depends on $\mathcal{X}_{t_n}$, i. e., for any $s_{n+1} \in S$:

$$Prob\left(\mathcal{X}_{t_{n+1}} \leq s_{n+1} \mid \mathcal{X}_{t_n} = s_n\right) = Prob\left(\mathcal{X}_{t_{n+1}} \leq s_{n+1} \mid \mathcal{X}_{t_0} = s_0, \ldots, \mathcal{X}_{t_n} = s_n\right). \tag{1.1}$$

This equation is generally known as the *Markov property*. Most often, Markov processes used for probabilistic model checking are invariant to time shifts, i. e., for any $t, t' \in \mathcal{T}$, such that $t' > t$, and $s', s \in S$ we have:

$$Prob\left(\mathcal{X}_{t'} \leq s \mid \mathcal{X}_t = s'\right) = Prob\left(\mathcal{X}_{(t'-t)} \leq s \mid \mathcal{X}_0 = s'\right) \tag{1.2}$$

In this case we have a *time-homogeneous Markov process* for which the next state only depends on the current state but neither on the previous states nor on how long we have been already in the current state.

In this thesis, we consider a Markov chain to be a time-homogeneous Markov process with the discrete state space S and the index set $\mathcal{T} = \mathbb{R}_{\geq 0}$ for continuous time or $\mathcal{T} = \mathbb{N}_{\geq 0}$ for discrete time. Moreover, unless stated otherwise, we assume a finite state space $S = \{1, \ldots, N\}$ with $|S| = N$. Conditions (1.1) and (1.2) mean that in a time-homogeneous Markov process, the state residence times must be random variables that have a memoryless distribution. The latter implies that the state residence times in a continuous-time Markov chain need to be exponentially distributed, and in a discrete-time Markov chain need to be geometrically distributed. Before we proceed with more details on discrete- and continuous-time Markov chains, we provide several useful definitions.

Definition 1 *A Markov chain is called* irreducible *if for any two states $s, s' \in S$ there exists $t \in \mathcal{T}$ such that $Prob\left(\mathcal{X}_t = s' \mid \mathcal{X}_0 = s\right) > 0$.*

Informally, *irreducible* means that every state is reachable from every other state.

Definition 2 *A state $s \in S$ of a Markov chain is called* absorbing *if for any $t \in \mathcal{T}$ and $s' \in S$ such that $s' \neq s$ we have $Prob\left(\mathcal{X}_t = s' \mid \mathcal{X}_0 = s\right) = 0$.*

Clearly, an absorbing state is a state from which there is a zero probability of exiting.

Definition 3 *A state $s \in S$ of a Markov chain is called* transient *if the following holds:*

$$\lim_{t \to \infty} \left(Prob\left(\mathcal{X}_t = s \mid \mathcal{X}_0 = s\right)\right) = 0.$$

The limit above states that, the probability to return to the transient state with time going to infinity is zero.

Definition 4 *A Markov chain is called* absorbing *if for any non-absorbing state $s \in S$ there exists an absorbing state $s' \in S$ and $t \in \mathcal{T}$ such that $Prob\left(\mathcal{X}_t = s' \mid \mathcal{X}_0 = s\right) > 0$.*

Now, with the main definitions introduced we proceed with the formal representation of continuous- and discrete-time Markov chains. We will also explain the ways of computing their transient and stationary probabilities.

1.1.1 Discrete-time Markov chains

Below we give a formal definition of a discrete-time Markov chain (*DTMC*), introduce the transient and stationary probabilities of the DTMC, and talk about their computation. At the end, we concentrate on the *steady-state detection* technique that allows to increase efficiency when computing transient probabilities of the DTMC.

Definition 5 *Let AP be a fixed and finite set of* atomic propositions *then a (labelled) DTMC is a tuple* $\mathcal{D} = (S,\ \mathbf{P},\ L)$ *where* S *is a finite set of* states, $\mathbf{P} : S \times S \to [0,1]$ *is a* probability matrix *such that* $\sum_{s' \in S} \mathbf{P}(s,s') = 1$ *for all* $s \in S$*, and* $L : S \to 2^{AP}$ *is a* labeling *function which assigns to each state* $s \in S$ *the set* $L(s)$ *of atomic propositions that hold in* s*.*

The matrix entry $\mathbf{P}(s,s')$ denotes the probability to move from state s to state s' in one step. A path through the DTMC is a sequence of states $\sigma = s_0\, s_1\, s_2 \ldots$ with $\mathbf{P}(s_i, s_{i+1}) > 0$ for all i. Let $Path^{\mathcal{D}}$ denote the set of all paths in the DTMC $\mathcal{D}$, then for any $\sigma \in Path^{\mathcal{D}}$ we define $\sigma[i]$ to be the $(i{+}1)$th state of σ, i. e., $\sigma[i] = s_i$. The probability space on $Path^{\mathcal{D}}$ can be defined using the standard Borel-space construction. Note that here we do not dwell upon the distinction between finite and infinite paths.

Transient probabilities

Let $\overrightarrow{p^o}$ be a row vector representing the initial-probability distribution of the DTMC, i. e., p^o_s denotes the probability to be initially in state s. Then the transient probabilities of the DTMC, with time $m \in \mathbb{N}$, are defined by the following recursive equation:

$$\overrightarrow{p^o\,(m)} = \overrightarrow{p^o\,(m{-}1)} \cdot \mathbf{P} \tag{1.3}$$

where $p^o_{s'}(m)$ is the probability to be in state $s' \in S$ at time m given the initial distribution vector $\overrightarrow{p^o\,(0)} = \overrightarrow{p^o}$.

Stationary probabilities

Definition 6 *The* limiting *state-probability*[1] *of the DTMC is a vector* $\overrightarrow{p^{o,*}}$ *such that:*

$$\overrightarrow{p^{o,*}} = \lim_{m \to \infty} \overrightarrow{p^o\,(m)} \tag{1.4}$$

Note that $p^{o,*}_s$ is the probability of being in the state s when taking a snapshot after a long time. Whenever the limit exists, it is also the solution of the following system of linear equations:

$$\overrightarrow{p} = \overrightarrow{p} \cdot \mathbf{P}, \quad \sum_{i \in S} p_i = 1 \tag{1.5}$$

In case the limit (1.4) does not exist, Equation (1.5) still has solutions. Note that in case of an irreducible DTMC, Equation (1.5) has a unique solution and otherwise

[1] The index "*" in Equation (1.4) will be used to distinguish between the exact probability values and their approximations introduced in Chapter 3.

infinitely many. The solution of Equation (1.5) is known as the *stationary* or *steady-state* distribution. It gives the proportion of time the DTMC spends in every state in the long run.

Before we proceed with Theorem 1 that states when the DTMC has unique limiting and steady-state distributions, we need to define the notion of an aperiodic DTMC. The latter is done using the notion of periodic states.

Definition 7 *A state s of the DTMC is called* periodic *if for some $d > 1$ it holds that for all $n \in \mathbb{N}_{\geq 1}$, such that $n \bmod d \neq 0$, the probability to return to the state s in n steps is 0.*

Definition 8 *The DTMC is called* periodic *if one of its states is periodic.*

Definition 9 *The DTMC is called* aperiodic *if it is not periodic.*

Note that a sufficient condition for an irreducible DTMC (cf. Definition 1) to be *aperiodic* is that there exists at least one state with a self loop.

Theorem 1 *[59] In an irreducible and aperiodic finite-state DTMC:*[2]

- *the limiting distribution $\overrightarrow{p^{o,*}}$ does exist*
- *$\overrightarrow{p^{o,*}}$ is independent of the initial distribution $\overrightarrow{p^o}$*
- *$\overrightarrow{p^{o,*}}$ is the unique steady-state distribution*

In order to determine the way of computing the steady-state probability of the DTMC let us note that, according to Equation (1.5), $\overrightarrow{p}$ is the left eigenvector of $\mathbf{P}$ that corresponds to the unit eigenvalue. As $\mathbf{P}$ is a stochastic matrix, the unit eigenvalue always exists, and no other eigenvalue exceeds it in modulus. Therefore, the steady-state probability can be computed as the dominant left eigenvector of the matrix $\mathbf{P}$. This computation can be done using the Power method described below.

Power method

This is a well-known numerical technique [131] for computing the dominant eigenvalue and its eigenvectors. In case of a stochastic matrix $\mathbf{P}$, it amounts to the following iterative procedure with $m \geq 1$ and $\overrightarrow{p^o(0)} = \overrightarrow{p^o}$ being an initial vector:

$$\overrightarrow{p^o(m)} = \overrightarrow{p^o(m-1)} \cdot \mathbf{P} \tag{1.6}$$

For an aperiodic $\mathbf{P}$, the convergence is guaranteed, if in addition $\mathbf{P}$ is irreducible then the result does not depend on $\overrightarrow{p^o}$. In the latter case there is only one eigenvector that corresponds to the unit eigenvalue (one steady-state distribution).

As any other iterative method, the Power method is expected to give results with some predefined error $\varepsilon > 0$. According to [131], the number K of iterations required to satisfy the error bound ε can be approximated by:

$$K = \frac{\log_2 \varepsilon}{\log_2 |\lambda_2|}$$

[2] Note that a finite-state DTMC is always positive recurrent [44].

where λ_2 is the sub-dominant eigenvalue of $\mathbf{P}$. In practice, however, λ_2 is difficult to compute and other convergence tests are used [131], such as:

1. An **absolute-convergence** test: $\left\| \overrightarrow{p^o\,(i)} - \overrightarrow{p^o\,(i+M)} \right\|_v < \varepsilon$
2. A **relative-convergence** test: $\max_{j \in \mathbb{N}_{[1,N]}} \left(\frac{|p_j^o(i+M) - p_j^o(i)|}{|p_j^o(i+M)|} \right) < \varepsilon$

In general the parameter $M > 0$ here is a function of the convergence rate and the iteration index i, but for simplicity M can be taken constant. Unfortunately, none of these convergence tests can guarantee the desired error bound because both of them are just the necessary conditions of convergence. Stewart [131] therefore suggests to envisage a *battery* of such convergence tests[3] all of which must be satisfied before the Power method result is accepted as being sufficiently accurate.

Steady-state detection

Notice that for an initial distribution $\overrightarrow{p^o}$, the *limiting* state-probability $\overrightarrow{p^{o,*}}$ of the DTMC is computed as a limit of the transient-probability vector $\overrightarrow{p^o\,(m)}$, that is recursively defined by Equation (1.3). Moreover, the Power method that allows to compute $\overrightarrow{p^{o,*}}$, see Equation (1.6), is nothing more than an iterative procedure for computing the limit (1.4) with the provided convergence tests aimed at detecting the limiting behavior.

Based on these observations, as it is suggested in [97], an optimization called *steady-state detection* can be applied when computing transient probabilities $\overrightarrow{p^o\,(m)}$ for large values of m. In essence, the idea of steady-state detection is that when computing $\overrightarrow{p^o\,(m)}$ we can stop iterating if the limiting probability is reached, i. e., $\overrightarrow{p^o\,(m)} = \overrightarrow{p^{o,*}}$. Since the probability distribution $\overrightarrow{p^{o,*}}$ is typically unknown, the approach boils down to applying the convergence tests of the Power method for detecting the limiting behavior at iteration m, provided the error bound ε is respected.

1.1.2 Continuous-time Markov chains

Below we give a formal definition of the continuous-time Markov chain (*CTMC*) and the embedded DTMC. Further we introduce the transient and stationary probabilities of the CTMC and talk about their computation.

Definition 10 *Let AP be a fixed and finite set of* atomic propositions *then a (labelled) CTMC is a tuple* $(S,\ \mathbf{Q},\ L)$ *where* S *is a finite set of* states, $L : S \to 2^{AP}$ *is a* labeling *function and* $\mathbf{Q} : S \times S \to \mathbb{R}$ *is a* generator matrix. *The elements of* $\mathbf{Q} = (q_{s,s'})$ *are such that for all* $s, s' \in S$ *and* $s \neq s'$ *we have* $q_{s,s'} \geq 0$, *and for all* $s \in S$ *we have* $q_{s,s} = -\sum_{s' \in S,\, s \neq s'} q_{s,s'}$.

The state-residence times of the CTMC are exponentially distributed. The value of $q_{s,s'}$ defines the rate of taking the transition from state s to s', and thus the time spent in state s is governed by the total exit rate $|q_{s,s}|$. On leaving the state s, a discrete

[3] There are other necessary conditions of convergence that are easy to check.

probabilistic choice takes place among the state successors, i. e., all $s' \in S$ for which $q_{s,s'} > 0$. The probability to move from state s to its successor s' is defined by the embedded DTMC.

Definition 11 *The embedded DTMC* $(S,\ \mathbf{P},\ L)$ *of a CTMC* $(S,\ \mathbf{Q},\ L)$ *is a discrete-time Markov chain such that for any* $s, s' \in S$ *we have:*

$$\mathbf{P}(s,s') = \begin{cases} q_{s,s'}/|q_{s,s}| & \text{if } s \neq s' \text{ and } |q_{s,s}| > 0 \\ 0 & \text{if } s \neq s' \text{ and } |q_{s,s}| = 0 \\ 0 & \text{if } s = s' \text{ and } |q_{s,s}| > 0 \\ 1 & \text{if } s = s' \text{ and } |q_{s,s}| = 0 \end{cases}$$

It is easy to see that the embedded DTMC does not have states with self-loops except for states $s \in S$ such that $|q_{s,s}| = 0$, i. e., the absorbing states.

Clearly, any CTMC can be represented as a tuple $(S,\ \mathbf{P}, E,\ L)$ where $(S,\ \mathbf{P},\ L)$ is the embedded DTMC and $E : S \to \mathbb{R}_{\geq 0}$ is such that $E(s) = |q_{s,s}|$, i. e., it provides the state *exit rates*. Using this representation, the probability of leaving the state s within t time units can be expressed as $1 - e^{-E(s)\cdot t}$, and the probability of taking the transition to state s' within time t as $\mathbf{P}(s,s') \cdot (1 - e^{-E(s)\cdot t})$.

A path through a CTMC is a sequence of states and sojourn times $\sigma = s_0\, t_0\, s_1\, t_1 \ldots$ with $\mathbf{P}(s_i, s_{i+1}) > 0$ and $t_i \in \mathbb{R}_{\geq 0}$ for all i. Let $Path^{\mathcal{C}}$ denote the set of all paths in the CTMC, then for any $\sigma \in Path^{\mathcal{C}}$ and $t \in \mathbb{R}_{\geq 0}$ we define $\sigma@t$ to be the state in σ occupied at time t. Formally, if $\sigma[i]$ is the $(i+1)$th state on the path σ, then $\sigma@t = \sigma[i]$ for the smallest index i such that $t \leq \sum_{j=0}^{i} t_i$. Note that once again the probability space on $Path^{\mathcal{C}}$ can be defined using the standard Borel-space construction; for details, see [8].

Transient probabilities

The transient probabilities of the CTMC are defined by the following differential equation:

$$\frac{d\overrightarrow{\pi^{o,*}(t)}}{dt} = \overrightarrow{\pi^{o,*}(t)} \cdot \mathbf{Q}, \tag{1.7}$$

where $\overrightarrow{\pi^{o,*}(t)}$ is a vector of state probabilities[4] after a delay of t time-units. In other words $\pi_i^{o,*}(t)$ is the probability to be in state i after t time-units. Provided with the initial distribution $\overrightarrow{p^o}$, the solution of Equation (1.7) is:

$$\overrightarrow{\pi^{o,*}(t)} = \overrightarrow{p^o} \cdot e^{\mathbf{Q}\cdot t} \tag{1.8}$$

The value of $\overrightarrow{\pi^{o,*}(t)}$ can be computed using numerical techniques such as Jensen's method, also known as uniformization.

Jensen's method (Uniformization) We first notice that for a real number q, called the *uniformization rate*, such that $q \geq \max_{i \in S} |q_{i,i}|$ the generator matrix of the CTMC can be represented as $\mathbf{Q} = q \cdot (\mathbf{P} - \mathcal{I})$. Here $\mathbf{P}$ is a stochastic matrix, called *uniformized CTMC*, and $\mathcal{I}$ is the identity matrix of cardinality $|S|$. Then, using the representation of

[4] The index "*" in Equation (1.7) is introduced for technical reasons.

Q in Equation (1.8) and expanding the matrix exponent according to Taylor-McLaurin, one obtains:

$$\overrightarrow{\pi^{o,*}(t)} = \sum_{i=0}^{\infty} \gamma_i(t) \cdot \overrightarrow{p^o(i)} \tag{1.9}$$

where $\gamma_i(t) = e^{-q \cdot t} \frac{(q \cdot t)^i}{i!}$ is the Poisson density function and $\overrightarrow{p^o(i)}$ is given by Equation (1.3).

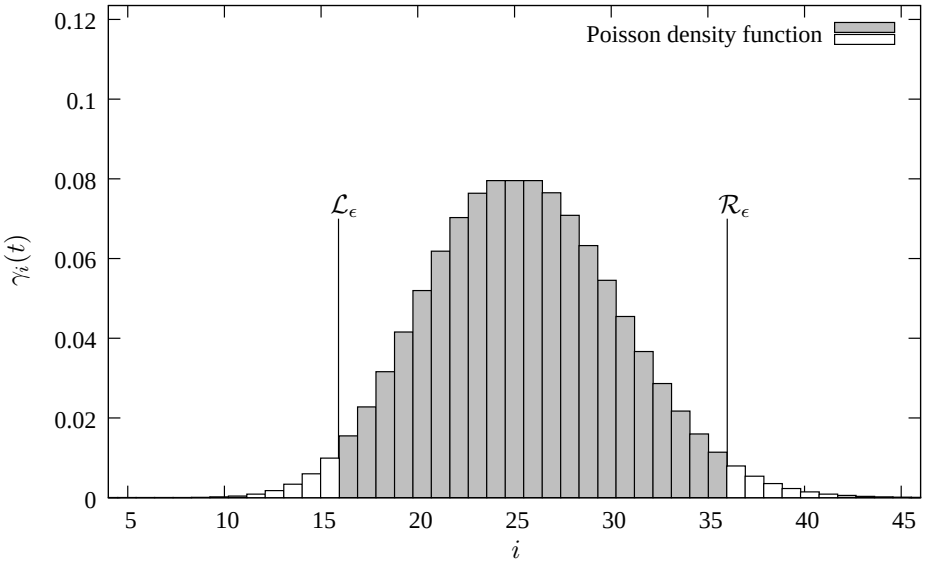


Figure 1.1: Poisson density function with $q \cdot t = 2$ and $\mathcal{R}_\epsilon$

The remarkable fact about Equation (1.9) comes from the particular shape of the Poisson density function (cf. Figure 1.1). Notice that for a given error bound $\varepsilon > 0$, the infinite sum can be truncated using the so-called left $\mathcal{L}_\epsilon$ and right $\mathcal{R}_\epsilon$ *truncation points* chosen in such a way that:

$$\sum_{i=0}^{\mathcal{L}_\epsilon - 1} \gamma_i(t) \leq \frac{\varepsilon}{2}, \text{ and } \sum_{i=\mathcal{R}_\epsilon + 1}^{\infty} \gamma_i(t) \leq \frac{\varepsilon}{2}.$$

The latter, since $\sum_{i=0}^{\infty} \gamma_i(t) = 1.0$, implies that $\sum_{i=\mathcal{L}_\epsilon}^{\mathcal{R}_\epsilon} \gamma_i(t) \geq 1 - \varepsilon$, allowing us to compute an ε-approximation of $\overrightarrow{\pi^{o,*}(t)}$ as:

$$\overrightarrow{\pi^{o}(t)} = \sum_{i=\mathcal{L}_\epsilon}^{\mathcal{R}_\epsilon} \gamma_i(t) \cdot \overrightarrow{p^o(i)}.$$

In practice, the computation of Poisson probabilities and truncation points for the approximation is typically done using the Fox-Glynn algorithm.

The Fox-Glynn algorithm For a real-valued function $f : \mathbb{N} \to \mathbb{R}$, the Fox-Glynn algorithm [50] allows for the following approximation:

$$\sum_{i=0}^{\infty} \gamma_i(t) f(i) \approx \frac{1}{W} \sum_{i=\mathcal{L}_\epsilon}^{\mathcal{R}_\epsilon} w_i(t) f(i),$$

where for all $i \in \mathbb{N}_{[\mathcal{L}_\epsilon,\, \mathcal{R}_\epsilon]}$ and some constant $\alpha \neq 0$ we have the *weights* $w_i(t) = \alpha\gamma_i(t)$ and the normalization weight $W = \sum_{i=\mathcal{L}_\epsilon}^{\mathcal{R}_\epsilon} w_i(t)$. Here $w_i(t)$ and W are used to prevent underflows during numerical computations. The following theorem gives the error bound for the approximation.

Proposition 2 *[50] For real-valued function f, and a Poisson density function $\gamma_i(t)$, if $\sum_{i=\mathcal{L}_\epsilon}^{\mathcal{R}_\epsilon} \gamma_i(t) \geq 1 - \frac{\varepsilon}{2}$ then the following holds:*

$$\left| \sum_{i=0}^{\infty} \gamma_i(t) f(i) - \frac{1}{W} \sum_{i=\mathcal{L}_\epsilon}^{\mathcal{R}_\epsilon} w_i(t) f(i) \right| \leq \varepsilon \cdot \|f\|,$$

where $\|f\| = \sup_{i \in \mathbb{N}} |f(i)|$.

More details on using the Fox-Glynn algorithm for computing $\overrightarrow{\pi^{o,*}(t)}$ can be found in Chapter 3.

Stationary probabilities

The stationary (steady-state) probabilities for the CTMC are a solution of the following system of linear equations:

$$\overrightarrow{p} \cdot \mathbf{Q} = \overrightarrow{0}, \quad \text{where} \quad \sum_{i \in S} p_i = 1. \tag{1.10}$$

The solution of this equation can be found by transforming it into the unit eigenvalue problem [130]: $\overrightarrow{p} \cdot \mathbf{P} = \overrightarrow{p}$, where $\mathbf{P}$ is the uniformized CTMC. Importantly to notice, it is well known [130] that if the uniformization rate q is chosen such that $q > \max_{i \in S} |q_{i,i}|$ then all eigenvalues of $\mathbf{P}$, except the unit eigenvalue, are strictly less than unity in modulus. The latter makes the embedded DTMC defined by $\mathbf{P}$ aperiodic and therefore ensures the existence of at least one steady-state probability distribution $\overrightarrow{p}$. Note that, the solution $\overrightarrow{p}$ is unique only if $\mathbf{P}$ is also irreducible.

1.2 Model checking Markov chains

Model checking is a technique that allows to check whether a system, represented as a model, satisfies its formal specification. The model is usually expressed as a directed graph which consists of nodes, edges and a set of atomic propositions associated with every node. The nodes correspond to system states, the edges represent possible transitions between the states, while the atomic propositions indicate the basic properties that hold at every particular state. The specification language, used to express system properties is typically some kind of temporal logic, e. g., Linear Time Logic (LTL) [114]

or Computation Tree Logic (CTL) [32]. With the system model and the specification in place, the model-checking problem can be expressed as follows: given a temporal-logic formula Ψ, a model M and the initial state s, decide if $M, s \models \Phi$. Since the model is typically clear from the context, further we omit M and simply write $s \models \Phi$.

In this section we discuss model-checking of systems that can be modeled as Markov chains. More specifically, we concentrate on model-checking techniques for discrete- and continuous-time Markov chains, see Sections 1.2.1 and 1.2.2, and their reward extensions, see Section 1.2.3. For DTMC and CTMC model checking we start with descriptions of corresponding temporal logics and then concentrate on the formal semantics and model-checking algorithms of their most interesting operators. We show how these model-checking procedures can be reduced to graph analysis and computing transient and stationary probabilities of Markov chains. Naturally, we pay more attention to the algorithms that are employed in the subsequent chapters of this dissertation and less to the ones that are not. More detailed information on model checking Markov chains can be found in Chapter 10 of the book "Principles of Model Checking" written by Baier & Katoen [14].

1.2.1 Model checking discrete-time Markov chains

Branching-time model checking of DTMCs was first introduced by Hansson and Jonsson in [56]. The approach allows for an automated verification of properties specified using Probabilistic Computation Tree Logic (PCTL) on a DTMC $\mathcal{D} = (S,\ \mathbf{P},\ L)$ with a set of atomic propositions AP. Below we introduce the PCTL syntax, semantics, and briefly discuss some of the model-checking procedures.

Using *state* formulas Φ and *path* formulas ϕ, the syntax of PCTL formulas can be inductively defined as follows:

$$\Phi ::= \mathit{true} \ \Big|\ a \ \Big|\ \Phi \wedge \Phi \ \Big|\ \neg \Phi \ \Big|\ \mathrm{L}_{\bowtie b}\,(\Phi) \ \Big|\ \mathrm{P}_{\bowtie b}\,(\phi)$$
$$\phi ::= \ \mathrm{X}\ \Phi \ \Big|\ \Phi\ \mathrm{U}^{[0,k]}\ \Phi \ \Big|\ \Phi\ \mathrm{U}\ \Phi.$$

Here, atomic proposition $a \in AP$, the probability bound $b \in [0, 1]$, $k \in \mathbb{N}$ represents discrete time and $\bowtie \in \{<, \leq, >, \geq\}$. Note that path formulas cannot be used on their own but only as part of a state formula. Also, every state formula Φ results in a set of states $\mathit{Sat}\,(\Phi) = \{\, s \in S \mid s \models \Phi \,\}$, the states that satisfy Φ. Therefore, when it is convenient, instead of a state formula we can use a set of states, e. g., we can write $\mathrm{L}_{\bowtie b}\,(\mathcal{G})$ for some $\mathcal{G} \subseteq S$.

Now, let us give an informal semantics of the main PCTL operators. The *long-run* operator $\mathrm{L}_{\bowtie b}\,(\Phi)$ asserts that the proportion of time spent in Φ-states in the long run meets the constraint $\bowtie\ b$. Note that this operator is not a part of the standard PCTL and was originally introduced in [4]. The *probability* operator $\mathrm{P}_{\bowtie b}\,(\phi)$ asserts that the probability measure of the paths satisfying ϕ meets the probability constraint $\bowtie\ b$. The *next* operator $\mathrm{X}\ \Phi$ asserts that a one-step transition is made to a Φ state. The *time-bounded until* operator $\Phi\ \mathrm{U}^{[0,k]}\ \Psi$ asserts that Ψ is satisfied at some (discrete) time instant in the interval $[0,\ k]$ and that at all preceding time instants Φ holds. The *unbounded-until* operator $\Phi\ \mathrm{U}\ \Psi$ is a variant of the time-bounded until where we take $k = \infty$. In this thesis, along with the operators described above, we will also use the following abbreviations: $\Diamond^{[0,k]}\Psi := \mathit{true}\ \mathrm{U}^{[0,k]}\ \Psi$ and $\Diamond\Psi := \mathit{true}\ \mathrm{U}\ \Psi$.

PCTL model checking is carried out in the same way as verifying CTL by recursively computing the set $Sat(\Phi)$. Further we present model-checking procedures for the time-bounded until, unbounded-until and long-run operators. These algorithms are important as they are going to be referenced in the subsequent chapters of this dissertation.

Time-bounded until operator. Following the informal semantics, we write that a path $\sigma \in Path^{\mathcal{D}}$ satisfies $\Phi\ \mathrm{U}^{[0,k]}\ \Psi$, i. e. $\sigma \models \Phi\ \mathrm{U}^{[0,k]}\ \Psi$, **iff** $\sigma[j] \models \Psi$ for some $j \leq k$, and $\sigma[i] \models \Phi$ for all $i < j$. Then, if $Path^{\mathcal{D}}(s)$ is a set of paths starting in state s, we write that $s \models \mathrm{P}_{\bowtie b}\left(\Phi\ \mathrm{U}^{[0,k]}\ \Psi\right)$ **iff** the probability measure $Prob\left(s,\ \Phi\ \mathrm{U}^{[0,k]}\ \Psi\right)$ of the set $\{\, \sigma \in Path^{\mathcal{D}}(s) \mid \sigma \models \Phi\ \mathrm{U}^{[0,k]}\ \Psi \,\}$ satisfies $\bowtie b$. A direct way to compute this probability is to find the least solution of the following linear equation system:

$$Prob\left(s,\ \Phi\ \mathrm{U}^{[0,k]}\ \Psi\right) = \begin{cases} 1 & \text{if } s \in S_1 \\ \sum\limits_{s' \in S} \mathbf{P}(s,s') \cdot Prob\left(s',\ \Phi\ \mathrm{U}^{[0,k-1]}\ \Psi\right) & \text{if } s \in S_? \wedge k > 0 \\ 0 & \text{otherwise} \end{cases}$$

where the sets S_1 and $S_?$ are defined as follows:

$$S_1 = \{\, s \mid s \models \Psi \,\},\ S_0 = \{\, s \mid s \models \neg\Phi \wedge \neg\Psi \,\},\ \text{and } S_? = S \setminus (S_1 \cup S_0). \tag{1.11}$$

One can simplify this system by replacing S_0 with

$$U_0 = S_0 \cup \{\, s \in S_? \mid \neg\exists \sigma \in Path^{\mathcal{D}}(s) : \sigma \models \Phi\ \mathrm{U}\ \Psi \,\}, \tag{1.12}$$

which can be found using a simple graph analysis in time $O(|S|+|\mathbf{P}|)$.

Alternatively, if the states $s \notin S_?$ are made absorbing, $Prob\left(s,\ \Phi\ \mathrm{U}^{[0,k]}\ \Psi\right)$ can be calculated using transient probabilities of the DTMC, cf. Section 1.1.1.

Definition 12 *For a DTMC $\mathcal{D} = (S,\ \mathbf{P},\ L)$ and $S' \subseteq S$, let $\mathcal{D}[S'] = (S,\ \mathbf{P}[S'],\ L)$ be the DTMC obtained by making all states in S' absorbing, i. e., $\mathbf{P}[S'](s,s) = \mathbf{P}(s,s')$ if $s \notin S'$ and otherwise $\mathbf{P}[S'](s,s') = 0$ for $s' \neq s$ and $\mathbf{P}[S'](s,s') = 1$ for $s' = s$.*

Let us consider the DTMC $\mathcal{D}[S \setminus S_?]$, then $Prob\left(s,\ \Phi\ \mathrm{U}^{[0,k]}\ \Psi\right)$ can be computed using the *forward-reachability* algorithm that employs transient probabilities of the DTMC:

$$Prob\left(s,\ \Phi\ \mathrm{U}^{[0,k]}\ \Psi\right) = \sum_{s' \in S_1} p^{o}_{s'}(k)\,. \tag{1.13}$$

Here $\overrightarrow{p^{o}(k)}$ is given by Equation (1.3) for which we should take $\overrightarrow{p^{o}} = \overrightarrow{1_{\{s\}}}$, i. e., the initial-distribution vector for starting in state s.

When doing model checking, we typically need to compute $Prob\left(s,\ \Phi\ \mathrm{U}^{[0,k]}\ \Psi\right)$ for all states $s \in S$. This can be done by employing the *backward-reachability* algorithm given by the following equation:

$$\overrightarrow{p(k)} = (\mathbf{P}[S_0 \cup S_1])^k \cdot \overrightarrow{1_{S_1}}. \tag{1.14}$$

Here $\overrightarrow{1_{S_1}}$ is the characteristic (column) vector of S_1 and $Prob\left(s,\ \Phi\ \mathrm{U}^{[0,k]}\ \Psi\right)$ is obtained as the s'th component of $\overrightarrow{p(k)}$. Note that the forward- and backward-reachability algorithms have the same time complexity.

Unbounded-until operator. We write that a path $\sigma \in Path^{\mathcal{D}}$ satisfies Φ U Ψ, i. e. $\sigma \models \Phi$ U Ψ, **iff** $\sigma[j] \models \Psi$ for some j, and $\sigma[i] \models \Phi$ for all $i < j$. Then $s \models \mathrm{P}_{\bowtie b}(\Phi \text{ U } \Psi)$ **iff** the probability measure $Prob(s,\ \Phi \text{ U } \Psi)$ of the set $\{\, \sigma \in Path^{\mathcal{D}}(s) \mid \sigma \models \Phi \text{ U } \Psi \,\}$ satisfies $\bowtie b$. It is easy to see that this probability can be computed using the same linear equation system as we have for the time-bounded until operator if we take $k = \infty$. In addition, one can simplify the equations by replacing S_1 with

$$U_1 = S_1 \cup \{\, s \in S_? \mid \forall \sigma \in Path^{\mathcal{D}}(s) : \sigma \models \Phi \text{ U } \Psi \,\}, \tag{1.15}$$

that can be found via a simple graph analysis in time $O(|S|+|\mathbf{P}|)$.

Long-run operator. Recall, that the proportion of time the DTMC spends in every state in the long run is defined by the steady-state distribution which is a solution of Equation (1.5), cf. Section 1.1.1. This distribution is unique, does not depend on the initial distribution, only if the DTMC is irreducible.

Keeping this in mind, the formal semantics of the long-run operator is given as follows. We write $s \models \mathrm{L}_{\bowtie b}(\Phi)$ **iff** the steady-state probability $Prob^{\infty}(s,\ \Phi)$ of being in the Φ states when starting in state s meets the constraint $\bowtie b$. $Prob^{\infty}(s,\ \Phi)$ can be computed using decomposition of the DTMC into its bottom strongly connected components.

Definition 13 *A* strongly connected component *(SCC) of a transition system is a maximal set of mutually reachable states. A bottom strongly connected component (BSCC) is an SCC from which no other SCC can be reached.*

For a DTMC each of its BSCCs can be seen as an irreducible subchain for which a unique steady-state distribution exists. Moreover, all states that do not belong to any BSCC are transient and thus $Prob^{\infty}(s,\ \Phi)$ is a combination of reachability and steady-state probabilities.

For the DTMC let $\{B_i\}_{i \in I}$ be a set of its BSCCs with the set of indexes I. For every BSCC B_i the steady-state distribution is computed by solving Equation (1.5). This, for any $s_i \in B_i$, allows us to obtain $Prob^{\infty}(s_i,\ Sat(\Phi) \cap B_i)$, i. e., the steady-state probability of being in the Φ states of BSCC B_i. Note that this probability is the same for all $s_i \in B_i$. The BSCC reachability probabilities $Prob(s,\ \Diamond B_i)$ are calculated using the techniques discussed earlier. As a result we get:

$$Prob^{\infty}(s,\ \Phi) = \sum_{i \in I} Prob(s,\ \Diamond B_i) \cdot Prob^{\infty}(s_i,\ Sat(\Phi) \cap B_i)\,.$$

1.2.2 Model checking continuous-time Markov chains

Model checking of CTMCs was first introduced by Aziz *et al.* in [5] and then refined by Baier *et. al.* in [8]. The approach allows for an automated verification of properties specified using Continuous Stochastic Logic (CSL) on a CTMC $\mathcal{C} = (S,\ \mathbf{Q},\ L)$ with a set of atomic propositions AP. Below we introduce the CSL syntax, semantics, and consider some of the model-checking procedures.

Similar to how it was done for PCTL, the syntax of CSL formulas can be inductively defined as follows:

$$\Phi ::= \mathit{true} \;\Big|\; a \;\Big|\; \Phi \wedge \Phi \;\Big|\; \neg\,\Phi \;\Big|\; \mathrm{S}_{\bowtie b}\,(\Phi) \;\Big|\; \mathrm{P}_{\bowtie b}\,(\phi)$$
$$\phi ::= \;\mathrm{X}\;\Phi \;\Big|\; \mathrm{X}^{[t_1,\,t_2]}\;\Phi \;\Big|\; \Phi\;\mathrm{U}^{[t_1,t_2]}\;\Phi \;\Big|\; \Phi\;\mathrm{U}\;\Phi.$$

Here we have an atomic proposition $a \in AP$, the probability bound $b \in [0,1]$, $t_1, t_2 \in \mathbb{R}_{\geq 0}$ (such that $t_1 \leq t_2$) represent time and $\bowtie \in \{<, \leq, >, \geq\}$.

CSL is a version of PCTL adapted for the continuous-time domain. The informal semantics of the newly introduced operators is as follows. The steady-state operator $\mathrm{S}_{\bowtie b}\,(\Phi)$ asserts that the steady-state probability of being in Φ states meets the boundary condition $\bowtie\ b$. The operator $\mathrm{X}^{[t_1,\,t_2]}\;\Phi$ is the timed variant of the next operator in PCTL; it asserts that a transition is made to a Φ state at some time $t \in [t_1,\,t_2]$. The time-interval until operator $\Phi\;\mathrm{U}^{[t_1,t_2]}\;\Psi$ is a generalization of the time-bounded until. It asserts that Ψ is satisfied at some time $t \in [t_1,\,t_2]$ and that at all preceding time instants Φ holds.

Further we discuss model-checking procedures for the time-interval until, unbounded-until and steady-state operators, because these algorithms are going to be referenced in the subsequent chapters of this thesis.

Time-interval until operator We write $s \models \mathrm{P}_{\bowtie b}\left(\Phi\;\mathrm{U}^{[t_1,t_2]}\;\Psi\right)$ **iff** the probability measure $\mathit{Prob}\left(s,\;\Phi\;\mathrm{U}^{[t_1,t_2]}\;\Psi\right)$ of the set of timed paths $\{\,\sigma \in \mathit{Path}^{\mathcal{C}}(s) \mid \sigma \models \Phi\;\mathrm{U}^{[t_1,t_2]}\;\Psi\,\}$ satisfies the constraint $\bowtie\ b$. For a path $\sigma \in \mathit{Path}^{\mathcal{C}}$ we write $\sigma \models \Phi\;\mathrm{U}^{[t_1,t_2]}\;\Psi$ **iff** there exists $t \in [t_1,\,t_2]$ such that $\sigma@t \in \mathit{Sat}\,(\Psi)$ and for all $t' < t$ we have $\sigma@t' \in \mathit{Sat}\,(\Phi)$.

Like for the time-bounded until of PCTL, see Section 1.2.1, the model-checking procedure for the time-interval until of CSL can be reduced to transient analysis, see Section 1.1.2. As before, we will use the sets $S_?$, S_0, S_1 and U_0 defined on page 12.

Definition 14 *For a CTMC $\mathcal{C} = (S,\,\mathbf{Q},\,L)$ and $S' \subseteq S$, let $\mathcal{C}[S'] = (S,\,\mathbf{Q}\,[S'],\,L)$ be the CTMC obtained by making all states in S' absorbing, i. e., for $\mathbf{Q}' = \mathbf{Q}\,[S']$ we have $q'_{i,j} = q_{i,j}$ if $i \notin S'$ and 0 otherwise.*

For simplicity, below we only consider the case of $t_1 = 0$, i. e., the time-bounded until formula $\Phi\;\mathrm{U}^{[0,t]}\;\Psi$. Given the CTMC $\mathbf{Q}\,[S \setminus S_?]$, the value of $\mathit{Prob}\left(s,\;\Phi\;\mathrm{U}^{[0,t]}\;\Psi\right)$ can be calculated in two ways. First, for any state $s \in S$, it can be obtained employing Algorithm 1 (*forward-reachability*), where $\overrightarrow{1_{\{s\}}}$ is the row vector defining the initial distribution for starting in state s. Second, the values of $\mathit{Prob}\left(s,\;\Phi\;\mathrm{U}^{[0,t]}\;\Psi\right)$ for all $s \in S$ can be computed at once [81] using Algorithm 2 (*backward-reachability*), where $\overrightarrow{1_{S_1}}$ is the characteristic (column) vector of S_1. Note that both algorithms have the same time complexity and that the matrix exponent can be computed numerically using uniformization. Also, one can optimize computations by replacing S_0 with U_0.

Unbounded-until operator We write that a path $\sigma \in \mathit{Path}^{\mathcal{C}}$ satisfies $\Phi\;\mathrm{U}\;\Psi$, i. e. $\sigma \models \Phi\;\mathrm{U}\;\Psi$, **iff** $\sigma@t \models \Psi$ for some t, and $\sigma@t' \models \Phi$ for all $t' < t$. Then $s \models \mathrm{P}_{\bowtie b}\,(\Phi\;\mathrm{U}\;\Psi)$ **iff** the probability measure $\mathit{Prob}\,(s,\;\Phi\;\mathrm{U}\;\Psi)$ of the set $\{\,\sigma \in \mathit{Path}^{\mathcal{C}}(s) \mid$

Algorithm 1 Computing $Prob\left(s,\ \Phi\ \mathrm{U}^{[0,t]}\ \Psi\right)$ in a "forward" manner

1: Determine $\mathbf{Q}\left[S \setminus S_?\right]$
2: Compute $\overrightarrow{\pi^{s,*}(t)} = \overrightarrow{1_{\{s\}}} \cdot e^{\mathbf{Q}[S\setminus S_?]t}$
3: Return $Prob\left(s,\ \mathcal{A}\ \mathrm{U}^{[0,t]}\ \mathcal{G}\right) = \sum_{s' \in Sat(\Psi)} \pi^{s,*}_{s'}(t)$

Algorithm 2 Computing $Prob\left(s,\ \Phi\ \mathrm{U}^{[0,t]}\ \Psi\right)$ in a "backward" manner

1: Determine $\mathbf{Q}\left[S \setminus S_?\right]$
2: Compute $\overrightarrow{\pi^{*}(t)} = e^{\mathbf{Q}[S\setminus S_?]t} \cdot \overrightarrow{1_{S_1}}$
3: Return $\forall s \in S : Prob\left(s,\ \mathcal{A}\ \mathrm{U}^{[0,t]}\ \mathcal{G}\right) = \pi^{*}_{s}(t)$

$\sigma \models \Phi\ \mathrm{U}\ \Psi\}$, satisfies $\bowtie\ b$. Clearly, $Prob\left(s,\ \Phi\ \mathrm{U}\ \Psi\right)$ does not depend on time and therefore is computed using the embedded DTMC following the algorithms given in Section 1.2.1.

Steady-state operator We write $s \models \mathrm{S}_{\bowtie b}(\Phi)$ **iff** the steady-state probability to be in a Φ-state, when starting in state s, i. e., $Prob^{\infty}\left(s,\ \Phi\right)$, satisfies the constraint $\bowtie\ b$. The steady-state distribution of the CTMC is a solution of Equation 1.10, cf. Section 1.1.2, and is unique if the CTMC is irreducible. Therefore, $Prob^{\infty}\left(s,\ \Phi\right)$ is computed on the uniformized CTMC using the model-checking procedure of the long-run operator.

1.2.3 Model checking Markov reward models

As we know, the model-checking algorithms for DTMCs and CTMCs rely on well-developed standard numerical algorithms. Recently, the further work in this area has focussed on DTMCs and CTMCs decorated with rewards. The former are then called discrete time Markov reward models (DMRMs) and the latter continuous-time Markov reward models (CMRMs). The properties for these models can be specified using the reward extensions of PCTL and CSL, namely PRCTL [4] and CSRL [11].

PRCTL extends PCTL with operators to reason about long-run average, and more importantly, by operators that allow to specify constraints on (i) the expected reward rate at a time instant, (ii) the long-run expected reward rate per time unit, (iii) the cumulated reward rate at a time instant—all for a specified set of states—and (iv) the cumulated reward over a time interval. PRCTL allows to specify non-trivial, though interesting, constraints such as *"the probability to reach one of the goal states (via indicated allowed states) within n steps, while having earned an accumulated reward that does not exceed r, is larger than* 0.92*"*. Some example properties that can be expressed in PRCTL are:

- $\mathrm{P}_{\geq 0.3}\left(a\ \mathrm{U}^{[0,3]}_{[23,47]}\ b\right)$ – the probability that a b-state can be reached via a-states within 3 time units, while accumulating reward from 23 to 47, is at least 0.3.
- $\mathcal{Y}^{3}_{[3,5]}a$ – the accumulated reward rate in a-states, expected within 3 hops, is from 3 to 5.

DTMC	Synchronous Leader Election Protocol
	Birth-Death process
	Randomized Mutual exclusion
	Crowds Protocol
CTMC	Tandem Queuing Network
	Cyclic Server Polling System
	Wireless Group Communication Protocol
	Simple Peer-To-Peer Protocol
	Workstation Cluster

Table 1.1: The case studies

CSRL extends CSL with time- and reward-interval next and until operators. This allows one to express a rich spectrum of properties, for example:

- $\mathrm{P}_{\leq 0.5}\left(\mathrm{X}^{[0,2]}_{[10,\infty)}\, c\right)$ – the probability that a transition to a c-state can be made at time $t \in [0,2]$, with the reward accumulated until time t lying in $(10,\infty)$, is at most 0.5.
- $\mathrm{P}_{\geq 0.3}\left(a\ \mathrm{U}^{[0,3]}_{[23,47]}\ b\right)$ – has the same meaning as in case of PRCTL, but deals with continuous time.

Note that, as PCTL (CSL) is a sub-logic of PRCTL (CSRL), we are dealing with orthogonal extensions: anything that could be specified in PCTL (CSL) can be specified in PRCTL (CSRL), and more.

1.3 Case studies

In this section we present case studies that are used for our experiments throughout the thesis. Most of the provided systems come from industry and all of them can be modeled either as discrete- or continuous-time Markov chains, cf. Table 1.1. Here we present the top-level descriptions of the case studies because all the necessary details can be found in the referenced material. For our experiments we take the formal specifications of the models, as can be consumed by the probabilistic model checkers discussed in the next section, that are available for a free download from [115] and [105]. For compatibility reasons, the model parameters such as rates and probabilities are kept intact. Therefore we present their values only if an additional insight into the case study is required.

1.3.1 Synchronous Leader Election Protocol (SLE)

Synchronous Leader Election Protocol [76] (see also [94, 54, 48]) solves the following problem: Given a synchronous ring of N processors design a protocol such that they

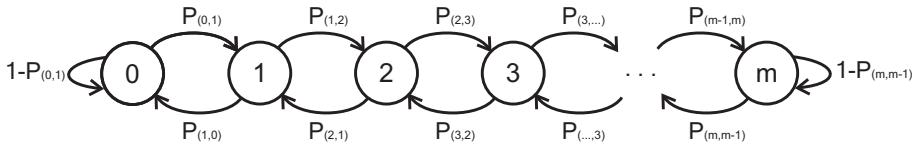


Figure 1.2: A birth-death process

will be able to elect a leader (a uniquely designated processor) by sending messages around the unidirectional ring.

The protocol proceeds in rounds where each round begins with all processors independently and uniformly choosing a random number (*an id*) from the set $[1 \dots K]$, with some predefined $K > 0$. The processors then pass their ids around the ring. If there is a unique id, then the processor with the maximum unique id is elected to be the leader, otherwise a new round begins. It is assumed that the ring is synchronous, i. e. there is a global clock. At every time slot a processor reads a message that was sent at the previous time slot (if it exists), makes at most one state transition, and then may send at most one message.

The typical properties verified for this case study are:

- $\mathrm{P}_{\leq q}\left(\Diamond^{[0,(N+1)\cdot 3]}\, \mathit{elected}\right)$ – the probability to elect a leader within N rounds is at most q.
- $\mathrm{P}_{\geq 1}\left(\Diamond\, \mathit{elected}\right)$ – eventually a leader is elected with probability one.

1.3.2 Birth-Death Process (BDP)

Birth-death processes [103, 80] are used in numerous fields, for instance to model the growth of a population. States in a birth-death process are numbered by integers that denote the current population size. In a birth-death process, the change in population size can occur by at most one, an increase in size is denoted as *"birth"* whereas a decrease is denoted as *"death"*. The birth-death processes are related to queuing theory, for example we might state that the population represents *"customers in the queue at the post office"*. Birth would then represent the arrival of a new customer and death the departure of a customer. An example of a finite birth-death process is depicted in Figure 1.2.

The finite Markov chain is obtained by limiting the maximum population size M. The probability of growth $P_{(N,N+1)}$ and death $P_{(N,N-1)}$ is made dependent on the current population size N as follows:

$$P_{i,j} = \begin{cases} \lambda & i = 0 \ \wedge \ j = 1 \ \wedge \ N = 0 \ \text{, birth from the initial state} \\ \frac{\lambda}{\lambda + (N \cdot \mu)} & j = i + 1 \ \wedge \ (0 < N < M) \ \text{, birth} \\ \frac{N \cdot \mu}{\lambda + (N \cdot \mu)} & j = i - 1 \ \wedge \ (0 < N < M) \ \text{, death} \\ \mu & i = M \ \text{, death from the } N = M \text{ state} \\ 0 & \text{, otherwise} \end{cases} \tag{1.16}$$

The constants λ and μ in Formula 1.16 are set to 0.8 and 0.001 respectively. In addition we define the probabilities of staying in the states 0 and M as $1 - \lambda$ and $1 - \mu$.

The typical properties verified for this case study are:

- $\mathrm{P}_{\geq q}\left(\mathrm{P}_{\geq p}\left(\Diamond^{[0,T]}(N = M)\right)\ \mathrm{U}\ (N = X)\right)$ – the probability to reach the population size X, if prior to that the probability of reaching the maximum population size within T steps remains $\geq p$, is at least q.
- $\mathrm{P}_{\geq q}\left(\Diamond(N = X)\right)$ – the probability to reach the population size X is at least q.

1.3.3 Randomized Mutual Exclusion (RME)

This case study is based on Pnueli and Zuck's solution [113] to the well-known mutual exclusion problem. In this algorithm, N processes $P_1 \ldots P_N$ make random choices based on coin tosses to ensure that they can enter their critical sections eventually, although not simultaneously. The processes can coordinate their activities by use of a common resource. The solution guarantees that at any time t there is at most one process in the critical-section phase and that every process can eventually enter the critical section. The model of the randomized mutual exclusion case study is rather complex, e. g. every process has 16 various states, and therefore we do not present any further details.

The typical property verified for this case study is:

- $\mathrm{P}_{\leq q}\left(\bigwedge_{j\neq 1}^{N} \neg \mathit{enter}_j\ \mathrm{U}\ \mathit{enter}_1\right)$ – the probability that the process P_1 is the first to enter the critical section is at most q.

1.3.4 Crowds Protocol (CP)

This protocol was developed by Reiter and Rubin [119, 115] to provide users with a mechanism for anonymous Web browsing. It uses random routing to hide each user's communications by directing their messages randomly within a group of similar users (a crowd).

The model includes N honest and $N/5$ dishonest crowd members. The latter ones are chosen at random and are able to observe the immediately preceding member on the path. The path from a particular source to a particular destination is set up only once, when the first message is sent. This happens as follows. The sender randomly selects a crowd member and forwards the message to it. This member, with some probability, sends the message directly to the destination or to the next selected router. Routing paths are reconstructed once the crowd changes; the number of such new route establishments is R, and is an important parameter that influences the state-space size. The protocol is designed to provide anonymity for message senders, i. e., under a specific parameter evaluation it is guaranteed that the real sender is indistinguishable from the other crowd members.

The typical property verified for this case study is:

- $\mathrm{P}_{\leq q}\left(\Diamond\, \mathit{observe}\right)$ – the probability of detecting the sender's id is at most q.

1.3.5 Tandem Queuing Network (TQN)

This case study, taken from [66, 63, 145, 121, 140, 144], consists of two sequentially composed queues, cf. Figure 1.3, each of capacity N. Messages arrive at the first

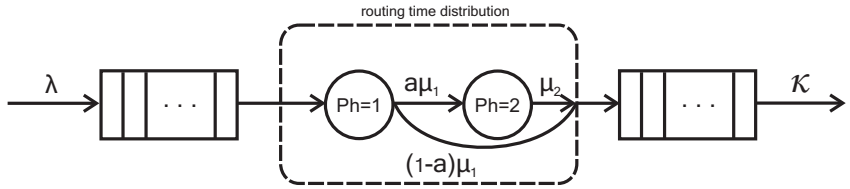


Figure 1.3: Tandem Queuing Network of two sequentially composed queues.

queue and stay in the queue for some time, before getting routed to the second queue, from where they eventually leave the system. The time between arrival of messages at the first queue is exponentially distributed with rate $\lambda = 4 \cdot N$. If the first queue is not empty and the second queue not full, then messages are routed from the first queue to the second queue. The routing-time distribution is a two-phase Coxian [36] distribution with parameters $\mu_1 = \mu_2 = 2$ and $a = 0.1$. The processing time at the second queue is exponentially distributed with rate $\kappa = 4$.

The typical properties verified for this case study are:

- $\mathrm{P}_{\leq q}\left(\mathit{true}\ \mathrm{U}^{[0,T]}\ \mathit{full}\right)$ – the probability that both queues become full within T time units is at most q.
- $\mathrm{P}_{\leq q}\left(\neg \mathit{full}_1\ \mathrm{U}\ \mathit{full}_2\right)$ – the probability that the second queue becomes full before the first queue is at most q.

1.3.6 Cyclic Server Polling System (CPS)

The case study describes a polling system [73, 141, 140, 63, 121, 145, 144] consisting of N equivalent stations and a server. Each station has a single-message buffer and the stations are attended by a single server in a cyclic order. The server starts by polling the first station. If this station has a message in its buffer (*busy*), the server starts serving the station. Once the station has been served, or if there was no message in the buffer (*idle*), the server start polling the next station. After polling all stations, the server returns to polling the first station and thus beginning a new cycle. The polling and service times are exponentially distributed with rates $\gamma = 200$ and $\mu = 1$. The arrival rate of messages at a station is equal for all stations and is exponentially distributed with rate $\lambda = \frac{\mu}{N}$.

The typical properties verified for this case study are:

- $\mathrm{P}_{\geq q}\left(\mathit{true}\ \mathrm{U}^{[0,T]}\ \mathit{busy}_1\right)$ – the probability that station 1 becomes busy within T time units is at least q.
- $\mathrm{P}_{\leq q}\left(\bigwedge_{j\neq 1}^{N} \neg \mathit{serve}_j\ \mathrm{U}\ \mathit{serve}_1\right)$ – the probability that the first station is served before any other station is at most q.
- $\mathit{busy}_1 \implies \mathrm{P}_{\geq q}\left(\Diamond^{[0,T]} \mathit{poll}_1\right)$ – if the first station is busy then the probability that it is served within T time units is at least q.

1.3.7 Wireless Group Communication Protocol (WGC)

WGC [102, 21, 100] is a variant of the centralized medium access protocol of the IEEE 802.11 standard for wireless local area networks. This protocol support real-time group communication between autonomous mobile stations and is centralized in the sense that the medium access is controlled by a fixed node in the network, the Access Point (AP).

The protocol communications go as follows. The AP polls the wireless stations, and on receipt of a poll message, stations may broadcast a message. Stations acknowledge the receipt of a message such that the AP is able to detect whether or not all stations have correctly received the broadcast message. In case of a detected loss, a retransmission by the originator takes place. It is assumed that the number of consecutive losses of the same message is bounded by a fixed constant OD, the so-called omission degree. This all refers to the transmission of time-critical messages; other messages are sent in another phase of the protocol. The AP controls these phases which are of a fixed duration.

In our study we consider 4 wireless stations and alter the state-space size by changing the omission degree OD. For simplicity we use the *fading model* of the system discussed in [100].

The typical property verified for this case study is:

- $\mathrm{P}_{\leq q}\left(\Diamond^{[0,24000]} \mathit{fail}\right)$ – the probability that a message originated by the AP is not received by at least one station within the duration of the time-critical phase ($t = 2.4$ milliseconds) is at most q.

1.3.8 Simple Peer-To-Peer Protocol (P2P)

This case study describes a simple peer-to-peer protocol [90] based on BitTorrent. A "torrent" is a small file which contains meta-data about the files to be shared and about the host computer that coordinates the file distribution. The model contains a set of clients trying to download a file that has been partitioned into K blocks. Initially, there is one client that has already obtained all of the blocks and N additional clients with no blocks. Each client can download a block (lasting an exponential delay) from any of the others. The time needed for downloading the block decreases as the number of computers possessing this block increases.

The typical property verified for this case study is:

- $\mathrm{P}_{\geq q}\left(\Diamond^{[0,T]} \mathit{done}\right)$ – the probability that all blocks are downloaded within T time units is at least q.

1.3.9 Workstation Cluster (WC)

This case study considers a dependable cluster of workstations and is originally proposed in [58], since then it has been used as a benchmark in various papers, e. g., [26, 143, 88, 116].

The cluster consists of two symmetric subsystems both consisting of N workstations. Inside a subsystem, the workstations are connected by means of switches that are connected by a backbone. Each component of the system (workstation, switch and backbone) is failure prone. There is a single repair unit that takes care of repairing

failed components. The failure and repair times are exponentially distributed. Depending on the number of operational and connected workstations, the system is said to offer maximum or minimum quality of service.

The typical properties verified for this case study are:

- $\mathrm{P}_{\geq q}\left(\Diamond^{[0,T]}\neg \mathit{minimum}\right)$ – the probability that a non-minimum service quality is provided within T time units is at least q.
- $\mathrm{S}_{\geq q}\left(\mathit{maximum}\right)$ – the probability that the maximum service quality is provided in the steady-state is at least q.

1.4 Probabilistic model checking tools

In this section we present several probabilistic model checkers, some of which support numerical model-checking techniques (e.g. PRISM, ETMCC) and some statistical model checking (e.g. YMER, VESTA). These tools are used throughout the thesis for comparison with the tool named MRMC that is introduced in Chapter 2.

1.4.1 PRISM

PRISM [88] stands for Probabilistic Symbolic Model Checker. The tool is developed in the Computing Laboratory at the University of Oxford, UK. The user interface of the tool is implemented in Java and the core algorithms are mostly developed in C++. PRISM supports three kinds of models: DTMC, CTMC and MDP. System models are described using the PRISM modeling language based on the Reactive Modules formalism of Alur and Henzinger [3]. Properties can be specified using PCTL (for DTMCs) or CSL (for CTMCs). There is also a limited support for the specification and analysis of properties based on costs and rewards.

For state space representation, PRISM offers a choice between MTBDDs[5], "sparse matrices" (PRISM^S) and "hybrid" data structures (PRISM^H). It is expected that the PRISM^S engine is faster, whereas PRISM^H should consume less memory. Regardless of the engine, PRISM always generates an MTBDD to represent the Markov chain. PRISM^S then generates a sparse matrix out of the MTBDD, depending on the kind of formula that is to be checked.

1.4.2 E $\vdash$MC^2

ETMCC, also known as E $\vdash$MC^2 [63] was developed by the Stochastic Modeling and Verification group at the University of Erlangen–Nürnberg, Germany, and the Formal Methods and Tools group at the University of Twente, the Netherlands. The development of the tool stopped in 2001 with the version 1.4.2. E $\vdash$MC^2 is written in Java and uses an explicit (i. e. not symbolic) data structure for state space representation, namely the sparse matrix. This model checker supports CTMC and DTMC models but does not use its own modeling language. Instead, it accepts models in (a subset of) the *.tra*-format as e.g. generated by the stochastic process algebra tool TIPPtool [61] and Petri net tool DaNAMiCS [29]. The state labeling with atomic propositions has to

[5]PRISM uses a modified version of the CUDD package [127].

be provided in a separate *.lab* file. It is also possible to use PRISM to generate these files directly from the PRISM modeling language. E $\vdash$MC^2 supports two temporal logics: CSL and aCSL. Just as CSL, aCSL provides means to reason about CTMCs, but unlike CSL, its basic constructors are actions instead of atomic state propositions; for details see [64].

1.4.3 Ymer

Ymer [141] is a command-line tool, written in C and C++, for verifying transient properties of stochastic systems. It is developed at Carnegie Mellon University, Pittsburgh, PA, United States. The tool supports generalized semi-Markov processes (GSMPs) [138], a superset of CTMCs. The language used for model specification is a subset of the PRISM language with several syntactic differences. Ymer implements statistical model checking for a subset of CSL. The algorithms are based on discrete event simulation [124] and sequential acceptance sampling [146], see also Part II of this dissertation. Ymer also supports numerical techniques, but the numerical engine is adopted from PRISM. The model checker offers a choice between either a simple or sequential acceptance sampling. There is also support for distributed acceptance sampling, meaning multiple machines can be used to generate samples independently.

1.4.4 VESTA

VESTA [121] is a Java-based tool for statistical analysis of probabilistic systems. It is developed at the University of Illinois at Urbana-Champaign, United States. It extends the statistical methods proposed in [146] and is based on Monte-Carlo simulation of the model and simple hypothesis testing [69]. Some modifications, such as support for the unbounded-until operator, were made to the original algorithms of [122]. VESTA supports two kinds of input models: DTMCs and CTMCs. The tool uses a Java-based language for their specification. A model description consists of sequential statements in combination with Java code. Each statement consists of a guard, rate and action. The language offers no explicit parallel composition. In addition to the Java-based language there is support for PMaude [1]. Verification properties can be specified using PCTL, CSL or QuaTEx [1].

1.5 Conclusion

In this chapter we introduced the basic concepts of probabilistic model checking. We started with presenting the two main formalisms used for modelling probabilistic systems, namely the discrete- and continuous-time Markov chain. Further, the temporal logics, such as PCTL and CSL were described, along with the most important model-checking algorithms. After that, an overview of the real-life systems that can be modeled as Markov chains was given. These systems are going to be used as case studies throughout this dissertation. In the end, we discussed the set of most known probabilistic model checkers, such as PRISM and Ymer. These tools implement model-checking techniques for DTMC and CTMCs allowing for an automated verification of PCTL and CSL properties.

In the next chapter we present a new model checker, named MRMC. After representing the main features of the tool we concentrate on its performance in comparison with the other tools. We provide various tool-implementation metrics and discuss the use of MRMC in various third-party projects.

Chapter 2

Markov Reward Model Checker

Nowadays, efficient algorithms for probabilistic model-checking of DTMCs, CTMCs and Markov decision processes are supported by several tools such as E $\vdash$MC^2 [63], PRISM [68], GreatSPN [17], VESTA [122], Ymer [141], and the APNN Toolbox [25]. Although these model checkers are able to handle a large set of measures of interest, the reward-based measures have received scant attention so far. Such measures are available in model checking of Markov *reward* models (MRMs). The latter are the underlying semantic model of various high-level performance modeling formalisms, such as reward extensions of stochastic process algebras, stochastic reward nets, and so on. For the recent advances in model checking of MRMs we refer to the PhD thesis of Lucia Cloth [34].

The tool presented in this chapter, named MRMC, supports the verification of MRMs, in particular DMRMs and CMRMs. The property-specification language for DMRMs is PRCTL and for CMRMs it is CSRL. For more details on these logics and the model-checking techniques we refer to Chapter 1.

MRMC is a command-line tool that has an easy input format and is realized in the C programming language, which allows the tool to be small and fast due to compiler-based optimizations and smart memory management within the implementation. Also, MRMC uses simple but high-performance data structures, such as: a slightly modified version of the well-known compressed-row, compressed-column representation of probability (rate) matrices, and bit vectors for representing sets of states. Since MRMC v1.2.2 the tool supports all major platforms, namely Microsoft Windows, Linux and Mac OS X. The tool is distributed under the GNU General Public License (GPL) [117] and is available for free download at [105]. In this chapter, unless stated otherwise, we will discuss MRMC v1.2.2.

The rest of the chapter is organized as follows. First, in Section 2.1 we introduce MRMC by means of a simple example and provide a high-level tool description. Section 2.2 is devoted to the implementation details, such as architectural solutions, data structures, low level algorithms etc. Further, in Section 2.4 we present an efficiency comparison between MRMC and several other probabilistic model checkers such as PRISM and VESTA. The MRMC implementation analysis and metrics are provided

in Sections 2.5 and 2.6 that include but are not limited to performance profiling and code complexity analysis. Section 2.7 describes the MRMC test suite and, among other things, discusses the provided test coverage. The use of MRMC in third-party projects is given in Section 2.8.

The results of this chapter are partially published as [84] and [78].

2.1 Functionality

In this section we provide an insight in the model-checking problems that can be solved by MRMC, and the model-checking algorithms it implements to allow for this functionality. We start with an example problem that requires computing a reward-based measure. Then, we show how the informal problem description can be transformed into the formal MRM model and the logical formula that has to be verified. We continue this section by providing the list of temporal logics and model-checking algorithms that are supported by MRMC. In addition, we describe some improvements of the standard verification procedures that have been realized in MRMC.

Example 1 *Consider a dice with only four wedges that have numbers* 1, 2, 3 *and* 4 *imprinted on them. Let the dice be biased in such a way that we get the beforementioned outcomes with probabilities* 0.4, 0.3, 0.2 *and* 0.1 *respectively. One can now play a simple game where the game round consists of continuously tossing the dice until winning, if the outcome is* 4 *and the accumulated outcome is from* 5 *to* 50*, or losing, if the outcome is* 1*. A natural question that can rise while playing this game is: "Is the probability to win this game, e. g. within* 100 *tosses, larger than* 0.5*?"*

The answer the question posed in Example 1 can be given if the described game is transformed into a DMRM model and the question is reformulated in terms of the PRCTL logic.

Example 2 *Let us consider Figure 2.1 that provides the formal DMRM model of the game described in Example 1. The model consists of five states where state* 1 *represents the moment at which the dice is tossed and states from* 2 *to* 5 *correspond to the dice outcomes from* 1 *to* 4*. These outcomes are transformed into state rewards and placed next to the states in the square braces. The* loss *and* goal *states are marked by labels enclosed in the curly braces. The* goal *label corresponds to the outcome* 4*. Recall that in order to win, by reaching the state* 4*, the accumulated outcome has to be within* 5 *and* 50*.*

Considering the DMRM model in Figure 2.1, the question posed in Example 1 can be formulated as the following PRCTL formula: $\mathrm{P}_{>0.5}\left(\neg \mathit{loss}\ \mathrm{U}^{[0,199]}_{[5,50]}\ \mathit{goal}\right)$*. This property asserts that the probability to reach the goal state, without visiting the loss state within* 199 *time steps, and the accumulated reward being from* 5 *to* 50*, is larger than* 0.5*. Notice that we have the upper time bound* 199 *that in the model corresponds to* 100 *dice tosses.*

Example 2 provides a typical model-checking problem that can be solved using MRMC. Moreover, the tool supports verification of four types of temporal logics, namely: PCTL, CSL, PRCTL and CSRL. Further we outline the model-checking algorithms implemented in MRMC logic wise, and then indicate the realized extensions of these algorithms that are not bound to a particular kind of logic.

For PCTL the realized algorithms are mostly discussed by Hansson and Jonsson in [56]. The exception is a long-run operator which is handled similar to the steady-state operator of CSL. The supported algorithms for PRCTL have been described by Andova *et al.* [4]. Model-checking techniques for CSL are derived from [8] and for its reward extension CSRL from [33] (see also [11, 57]). For the latter one we have implemented two algorithms for time- and reward- bounded until formulae. One is based on discretization [133] and another on uniformization and path truncation [118]. The algorithms for PRCTL and CSRL support both state and impulse rewards. It is important to note that the model-checking procedures integrated in MRMC were complemented with the following extensions that are aimed at improving the tool's performance and accuracy:

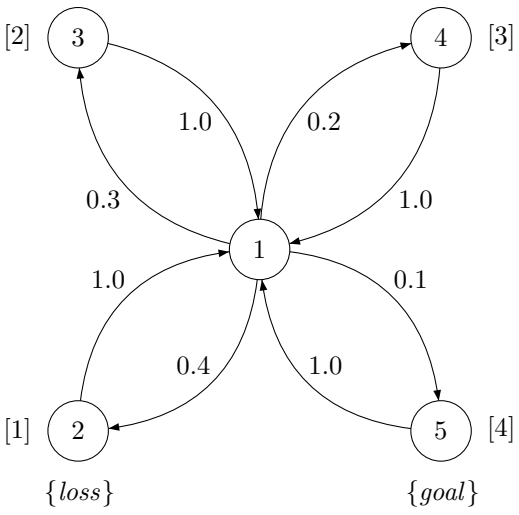


Figure 2.1: The dice game: DMRM model

Steady-state (long-run) operator of CSL (PCTL). For the operator $S_{\bowtie b}(\Psi)$ the algorithmic improvement lies with searching only for BSCCs that can contain Ψ states, as opposed to searching for all BSCCs. The modification that was done to the model-checking algorithms is straightforward and therefore we do not explain it in further details.

Unbounded-until operator of CSL (PCTL). For model checking $P_{\bowtie b}(\Phi \text{ U } \Psi)$, we first exclude states, using graph reachability analysis, from which Ψ states are always or never reachable. Then the model checking procedure for the remaining states is carried out as usual. All techniques required for this improvement are described in [31].

Time-bounded until operator of CSL. We have implemented a uniformization procedure [8] with a precise on-the-fly steady-state detection which is discussed in Chapter 3. Similar to unbounded-until operator, the technique of [31] is employed to detect and remove states from which the Ψ states are never reached. Also we employ ideas, described in [81], that allow to compute the reachability probabilities for all initial states at once.

Bisimulation minimization. The bisimulation minimization algorithms have been realized for PCTL, CSL, PRCTL and CSRL, in the latter two cases without impulse rewards. For more details consider Chapter 4.

In the next section we discuss the MRMC implementation in more details.

2.2 Implementation details

Since all the main algorithms integrated in MRMC are referenced in the previous section, in order to give a better insight into the implementation, below we concentrate on its details such as the architecture, realized data structures and low-level algorithms.

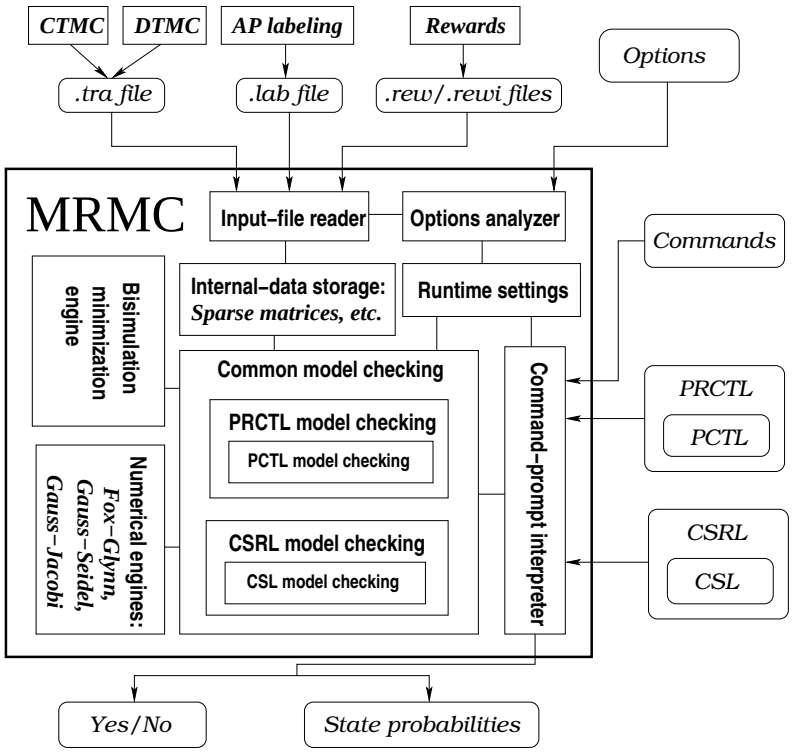


Figure 2.2: Tool architecture of MRMC

A sketch of the MRMC tool architecture is provided in Figure 2.2. Considering this figure, we present a brief textual description of the main MRMC components. Their correspondence to the source files can be found in Table A.1 of Appendix A.2. Note that in this section we do not discuss interactions between the MRMC components and the tool usage. These are illustrated in Section 2.3 by means of several examples.

"Options analyzer" – is responsible for parsing the command-line options of MRMC. It invokes reading of the input files and sets the run-time parameters of the tool, such as the logic we use to specify properties and/or the use of formula-dependent/independent lumping (see Chapter 4).

"Runtime settings" – stores the run-time settings of MRMC, for example the error bounds, the maximum number of iterations for the numerical methods and the result-output formats.

"Input-file reader" – is responsible for reading the `.tra`, `.lab`, `.rew` and `.rewi` files that are used to specify the input MRM model. For more details on the input-file formats we refer to Section 2.3.

"Internal data storage" – contains implementations of various data structures used in MRMC, among which are a sparse matrix, a bit set, structures for storing state labels, and splay trees used in bisimulation minimization algorithms.

"Command-prompt interpreter" – is based on Yacc and Lex [74] and responsible for: *(i)* interpreting the MRMC shell commands (setting error bounds, desired numerical methods etc.), and PRCTL (PCTL) or CSRL (CSL) properties, *(ii)* controlling the bottom-up recursive ascend over the parse tree of the logical formula, and *(iii)* printing out the model checking results. The details about syntax and semantics of available commands can be found in [106].

"Common model checking" – contains a set of generally-used algorithms applied in model checking, e.g. procedures for searching BSCCs, and encloses the algorithms for model checking PRCTL (PCTL), CSRL (CSL).

"Bisimulation engine" – provides algorithms for lumping state spaces of input models, which are labeled DTMCs, CTMCs and their reward extensions.

"Numerical engines" – contains implementations of numerical methods, such as Fox-Glynn algorithm for computing Poisson probabilities, Gauss-Seidel and Gauss-Jacobi iterative methods for solving systems of linear equations.

2.2.1 Data structures

Below we discuss and substantiate the choices of the main data structures used in MRMC.

Sparse matrices Storing a Markov chain may be quite a challenge, since most real-life models are represented by chains with millions of states and transitions. Fortunately, most transition matrices that appear in probabilistic model checking have a very sparse structure, i. e. contain a large number of zeroes. Therefore using sparse-matrices, such as a compressed-row (compressed-column) representation etc. (see [112] for more details), as an internal repository for probability (rate) matrices is advantageous. These structures allow to avoid the storage of, and computation on, a large number of zeros while keeping the manipulations with data relatively cheap.

For MRMC, as recommended in [131], we have chosen the compressed-row representation because it assures a high efficiency of matrix-vector multiplications which are at the core of numerical model checking. Also, similar data structures were implemented in the (by the year 2004) fastest serial and parallel explicit Markov chain solver developed by Alexander Bell, for more details we refer to [16].

In our implementation the sparse matrix is represented by a structure containing a number of rows: **nrows**; the number of columns: **ncols**; an array that stores the number of non-zero off-diagonal elements for each row: **succ**; an array of pointers to the structure representing a matrix row: **rows**; and an array: **pred**, that contains the

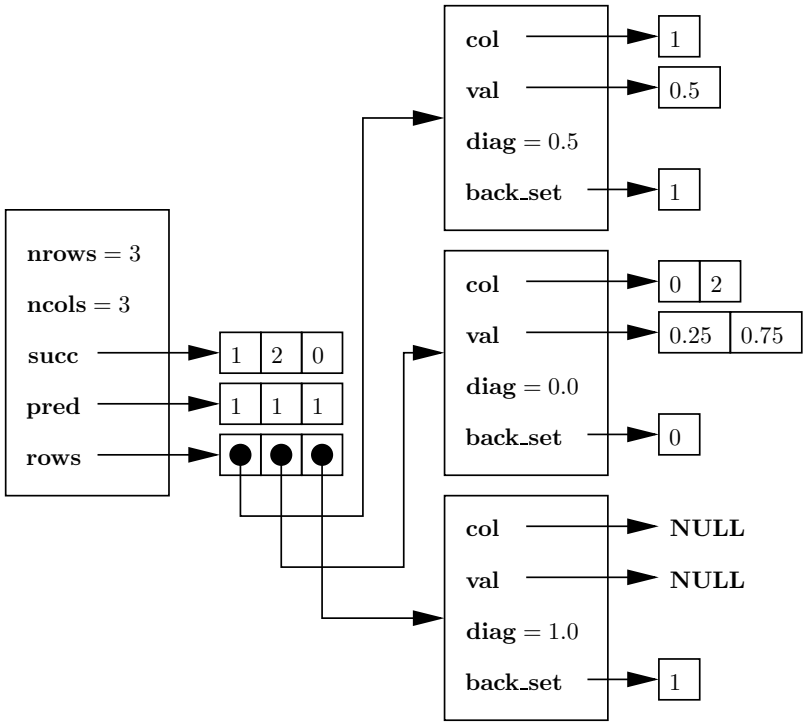


Figure 2.3: An example of the sparse-matrix representation used in MRMC

number of predecessor states for every state in the transition matrix. Note that the self loops are not taken into account by **pred**.

The row structure has several fields, namely the diagonal element: **diag**; an array of non-zero off-diagonal values: **val**; an array of corresponding column indexes: **col**; and an array: **back_set**, that contains predecessors of the state corresponding to the given row index. The **back_set** array is used for the bisimulation-minimization procedure and in the model-checking algorithms of PCTL and CSL.

Example 3 *Consider the matrix* $\mathbf{P}$*:*

$$\mathbf{P} = \begin{bmatrix} 0.50 & 0.50 & 0.00 \\ 0.25 & 0.00 & 0.75 \\ 0.00 & 0.00 & 1.00 \end{bmatrix}$$

Although it is not really sparse, we can still use it for the illustrative purposes and explain how it is transformed into the compressed-row representation of MRMC. Note that in this example the state, matrix and array indices start with 0*.*

Figure 2.3 shows the data structures that are allocated for matrix $\mathbf{P}$ *in MRMC. The matrix structure (on the left of the figure) has the number of columns and rows set to* 3*. Its* **succ** *array contains values* 1, 2 *and* 0 *because the zero row has only one non-zero off-diagonal element* 0.5*, the first row has two elements* 0.25 *and* 0.75*, and the second row has none. The* **pred** *array contains ones because the predecessor of state* 0 *is state* 1*, of state* 1 *is state* 0*, and of state* 2 *is state* 1*.*

Now, let us take the zero row of the matrix **P**. *The pointer to the structure representing it is stored in the zero element of the array* **rows**. *The structure has the following values assigned to its fields: the array* **col** *consists of only one element, i. e. index* 1*, because the only non-zero off-diagonal element of the row is located in the column* 1. *The value of this element is stored in the corresponding element of the array* **val**. *The field* **diag** *is set to* 0.5*, since this is the matrix diagonal element located in the zero row. The array* **back_set** *contains the state* 1*, because state* 0 *has an incoming transition from state* 1.

An advantage of the compressed-row representation is that it gives an easy access to the matrix rows. The latter is crucial for the efficiency of matrix-vector multiplications, which are at the heart of the numerical model checking. Storing rows separately simplifies the procedure of making states absorbing. The fact that the matrix diagonal elements are stored apart from the non-diagonal elements facilitates optimizations of matrix transformations, such as computation of an embedded Markov chain.

There are many other different ways to store transition matrixes efficiently, among them are MTBDDs [51, 7], hybrid approaches [89, 91, 111] and many others. MTBDDs are known for a very compact state-space representation on models that exhibit a highly regular structure, but suffer from a non-ideal performance when used in model checking. The hybrid approaches are a trade off between an efficient state-space representation and a good performance in model-checking applications. In this respect, sparse matrices, although potentially not as efficient in memory usage as MTBDDs, allow for a good performance as is shown in Section 2.4.

Bit sets During model checking it is very often needed to manipulate sets of states. For example, while model checking a nested formula, sub-formulas have to be treated first, resulting in sets of states satisfying these formulas. Therefore a data structure, capable of efficiently representing sets of states, has to be implemented.

In MRMC we have chosen to use a so-called bit set. The idea behind it is relatively simple. Having a set of state indices to store, we allocate a bit vector, where bits with indices corresponding to our states, are set to one, and all other are set to zero. Operations on such bit sets are simple. For instance, the computation of the complement of a set amounts to applying an exclusive *or* operation to the given bit vector with the vector of the same size, filled with ones. We illustrate the bit-set structure implemented in MRMC with the following example.

Example 4 *Let* $S = \{2, 5, 9, 31, 62\}$. *In MRMC it is represented as a bit vector built of* 32*-bit sub-blocks, see Figure 2.4. Operations with this vector are simple and efficient, e.g. checking for the state* 62 *being in the set* S *amounts to accessing the sub-block number* $1 = floor(62/32)$ *and testing its bit number* $30 = mod(62, 32)$ *for having been set to one. Note that in this example the state and array indices start with* 0*, the function* $floor(.)$ *returns the integer part of the real value, and the function* $mod(.,.)$ *returns the remainder after dividing the first argument by the second one.*

Splay trees A splay tree is a self-balancing binary search tree with the additional unusual property that recently accessed elements are quick to access again. Splay trees are used in our implementation of the bisimulation minimization algorithms,

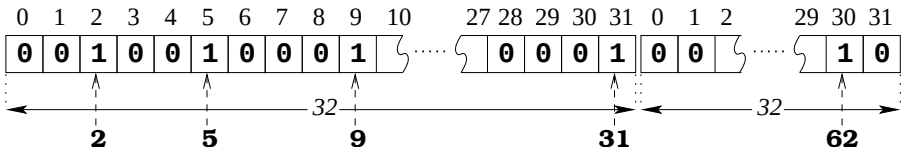


Figure 2.4: An example of the bit-set structure used in MRMC

for partitioning the state-space, to achieve a low time complexity [126]. In [40] it is suggested that using red-black trees, may be even more efficient. Our experiments with the red-black trees have shown that in general it is not the case. The latter is due to the much more complex implementation of the data structure and operations on it. Therefore, in MRMC we use an efficient implementation of splay trees developed by Daniel Sleator [125].

2.2.2 Basic algorithms

Although originated for other purposes, algorithms for carrying out graph analysis, solving linear equation systems and computing Poisson probabilities are heavily involved in the standard procedures of probabilistic model checking (cf. Chapter 1). MRMC implements several of such low-level algorithms and below we present the list of these algorithms along with the modifications we have done to them.

Searching for BSCCs. For model checking of the steady-state (long-run) properties of CSL (PCTL), it is essential to know the BSCCs of the considered Markov chain. To that purpose we have employed the Tarjan's algorithm [132] that looks for the maximum strongly connected components, and aimed it specifically at searching for BSCCs[1].

Gauss-Seidel and Gauss-Jacobi. These are well-known iterative methods used for solving systems of linear equations [131]. In MRMC they are utilized when model checking steady-state (long-run) as well as unbounded-until properties.

Fox-Glynn. The Fox-Glynn algorithm [50] is typically used for computing Poisson probabilities that arise when doing uniformization for time-bounded and time-interval until operators of CSL logic. We employ the method as explained in the original paper of Fox and Glynn, but apply improved error bounds. For more details see Chapter 3 of this thesis.

2.3 Tool usage

In this section we explaining the MRMC tool usage by discussing its inputs, outputs and presenting the experimental runs. We start with the list of tool input files and

[1]The modification that we have done to the original algorithm is fairly simple and therefore we do not discuss it in more details.

options, and then proceed with several examples. The latter are based on Example 2 and show how the formal DMRM model can be transformed into the MRMC input-file formats, how the tool can be started in the PRCTL mode, and how the verification of the PRCTL property can be done. In our examples we also reference the MRMC architecture, see Figure 2.2, in order to indicate the ways the main tool components interact with each other.

MRMC is a command line tool that provides a shell-like environment (*a command prompt*) where a user can specify the tool run-time options and the properties that have to be verified. On the start up, MRMC accepts several command-line options, e. g., that specify the logic (such as CSL or PRCTL), and expects four input files: a `.tra`-file describing the probability or rate matrix, a `.lab`-file indicating the state labeling with atomic propositions, a `.rew`-file specifying the state-reward structure, and a `.rewi`-file specifying the impulse-reward structure. For all supported logics the `.tra` and `.lab` files are compulsory, whereas `.rew` and `.rewi` files are used only for specifying MRMs. For more details on the command-line options and input files we refer to [106]. The following example illustrates the input-file formats of MRMC.

Example 5 *Let us consider the DMRM model of Example 2. This model can be seen as a superposition of three parts:* (i) *the DTMC given by state-transitions and corresponding distributions,* (ii) *the labeling function that maps sets of labels to the DTMC states, and* (iii) *the state-reward function that maps reward values to the DTMC states. In order to be used with MRMC, all these three parts have to be transformed into the MRMC input files. Such a translation is given in Table 2.1.*

The `game.tra` *file contains an intuitive text-based representation of the DTMC, i.e. its state transitions and corresponding probabilities. The* `game.lab` *file contains label declarations and maps sets of labels to the states of DTMC. Similarly the* `game.rew` *file contains mapping of the state rewards to the model states.*

Once the formal system model is translated into the MRMC input files it can be consumed by the tool as it is explained in the following example.

`game.tra`	`game.lab`	`game.rew`
STATES 5	#DECLARATION	2 1
TRANSITIONS 8	loss goal	3 2
1 2 0.4	#END	4 3
1 3 0.3	2 loss	5 4
1 4 0.2	5 goal	
1 5 0.1		
2 1 1.0		
3 1 1.0		
4 1 1.0		
5 1 1.0		

Table 2.1: The dice game: MRMC input files

Example 6 *In order to start MRMC with the input files given in Example 5 the following command should be executed in a shell environment such as* csh, bash *on Linux (Mac OS X), or* dos *on Microsoft Windows:*

```
MRMC/bin> mrmc prctl game.tra game.lab game.rewi
```

When executed, this command starts MRMC by triggering several of its components, see Figure 2.2. First "Options analyzer" parses the command-line arguments, setting up the PRCTL *logic as the current one in the "Runtime settings" component and invoking "Input-file reader" for processing the files* game.tra, game.lab *and* game.rewi. *At this stage "Internal-data storage" provides necessary data structures for storing the probability matrix, labeling and state rewards, which then become accessible through "Runtime settings". Once MRMC is started it produces the following output:*

```
 ------------------------------------------------------------------
|                    Markov Reward Model Checker                   |
|                        MRMC version 1.2.2                        |
|         Copyright (C) The University of Twente, 2004-2007.       |
|               Copyright (C) RWTH-Aachen, 2006-2007.              |
|                             Authors:                             |
|   Joost-Pieter Katoen (since 2004), Ivan S Zapreev (since 2004), |
|      Christina Jansen (since 2007), Tim Kemna (2005-2006),       |
|                  Maneesh Khattri (2004-2005)                     |
|          MRMC is distributed under the GPL conditions            |
|          (GPL stands for GNU General Public License)             |
|          The product comes with ABSOLUTELY NO WARRANTY.          |
| This is a free software, and you are welcome to redistribute it. |
 ------------------------------------------------------------------
Logic                    = PRCTL
Loading the 'game.tra' file, please wait.
States=5, Transitions=8
Loading the 'game.lab' file, please wait.
Loading the 'game.rew' file, please wait.
The Occupied Space is 992 Bytes.
Type 'help' to get help.
>>
```

where, first the MRMC logo is printed, then some general information about the accepted model and finally the MRMC shell invitation sign >>. *After that the tool is up and running, ready to accept user commands.*

When MRMC is started, the user gets access to the tools command prompt where (s)he can specify the tool run-time options, such as a use of certain algorithms, and the properties that have to be verified. For every verification problem the tool outputs a set of states that satisfy the given property and, if applicable, the list of probabilities. Note that the complete list of MRMC command-line options and command-prompt commands can be found in the tool documentation [106].

Example 7 *Extending Example 6, we can answer to the model checking problem of Example 1, by executing the following command in the MRMC command prompt:*

```
>>P{>0.5}[ !loss U[0,199][5,50] goal]
$RESULT: ( 0.0647999, 0.0000000, 0.0959998, 0.1199998, 0.1199997 )
$STATE: {  }
The Total Elapsed Model-Checking Time is 45 milli sec(s).
>>
```

By doing this we invoke the "Command-prompt interpreter" component that processes all commands of the MRMC shell. This component, using "Runtime settings" determines which model-checking engine is needed, in this case it is "PRCTL model checking", and then invokes it. As a result, we get two outputs: a probability vector `$RESULT`*, and a set of states* `$STATE`*. The former corresponds to the list of probabilities to satisfy the formula* $\neg loss\ \mathrm{U}^{[0,199]}_{[5,50]}\ goal$ *when starting in the first, second, etc. states. The latter one is the set of states in which the formula* $\mathrm{P}_{>0.5}\left(\neg loss\ \mathrm{U}^{[0,199]}_{[5,50]}\ goal\right)$ *is satisfied.*

Since, when playing the dice game, we always start in state 1*, i. e. we first toss the dice, from the vector* `$RESULT` *we can see that the probability to win the game within* 100 *dice tosses is just* `0.0647999` *and thus indeed* 1 *is not in the set* `$STATE`*.*

At this point we have described the main functionality of MRMC, referenced its model-checking algorithms, explained the choices for the internal data structures, and provided the examples of the tool usage. It is time to compare MRMC with the other state-of-the-art probabilistic model checkers. Since efficiency is one of the most important aspects of probabilistic model checking, the tool performance is of an utmost importance. Therefore, in the next section we experimentally compare performance of MRMC and the model-checking tools such as PRISM, Ymer, and etc.

2.4 Experiments and comparison

This section provides a comparative experimental study of MRMC and a substantial set of probabilistic model checkers, namely E $\vdash$MC^2, PRISM (sparse and hybrid mode), Ymer and VESTA, that are described in Section 1.4. We focus on fully probabilistic models, that is, finite-state DTMCs and CTMCs, and consider the temporal logics PCTL and CSL. The experiments are aimed at the verification time, i. e., the required time to verify a formula on a Markov chain, as well as peak memory usage, i. e., the maximal amount of memory needed during the verification. All experiments were carried out on a standard PC, and care was taken that equivalent input models are used. Since models, properties, testing environment, and tool settings are all publicly available, all reported experiments are repeatable and verifiable. The number of experiments carried out is substantial, and each experiment is repeated several times. In total, about 15,000 verification runs have been considered. The results of this section are published as [78], which presents a selection of the experiments from [110].

The versions and release dates of the considered tools are listed in Table 2.2. The five representative case studies are taken from the literature on performance evaluation and probabilistic model checking. There are three discrete-time and two continuous-time case studies. For each case study, we let the tools calculate the probability of some bounded and unbounded-until properties. In addition we included a nested property (with multiple until operators) in a discrete-time case study. In the continuous-time case studies, we also checked for steady-state properties, which is another important property type available in CSL (besides until properties). Table 2.3 presents the list of considered case studies (cf. Section 1.3), along with their minimal and maximal model sizes.[2]

[2]Unfortunately we were not able to generate larger state spaces for the SLE case study due to an error obtained from the CUDD package.

Tool name	Version	Release date
MRMC [a]	1.1.1b	March 2006
PRISM [a]	2.1	September 2004
E $\vdash$MC^2	1.4.2	2001
Ymer	3.0	February 2005
VESTA	2.0	September 2005

[a] This was the most recent version when we started our research.

Table 2.2: Tool versions used in efficiency comparison

timing	study	min/max, param.	# states	# transitions
discrete	SLE	min, $N = 4, K = 2$	55	70
		max, $N = 8, K = 4$	458,847	524,382
	BDP	min, $M = 100$	101	202
		max, $M = 100{,}000$	100,001	200,002
continuous	TQN	min, $N = 2$	15	23
		max, $N = 1023$	2,096,128	7,328,771
	CPS	min, $N = 3$	36	84
		max, $N = 18$	7,077,888	69,599,232

Table 2.3: Minimal and maximal model sizes per case study

2.4.1 Experimental setup

Below we describe the details of our experiments measuring the verification time and peak memory usage of the various tools. To give our conclusions a solid scientific basis, the experiment design was guided by the following principles:

- *Repeatability and Verifiability:* Every one should be able to repeat and verify our experiments; this is achieved by the fact that our models, properties, scripts and tool settings are publicly available.
- *Statistical Significance:* This has been achieved by repeating experiments several times and computing the standard deviation.
- *Encapsulation:* Our experiments should measure what we claim to measure (i. e. model check times and memory usage), no other influences. This has been achieved by carefully measuring the time and memory usage of the processes (see below) and by using a dedicated machine, thus the effect of disturbing factors such as network traffic, background processes is avoided.

Moreover, we have considered the tools as black boxes. That is, we have executed the tools, but not changed their source code[3]. Also, we chose the verification parameters (e. g. the algorithm for solving matrix equations) to be the same across all tools. For details on the models and measurements, we refer to [110].

Software and hardware settings. All experiments were performed on a standard PC with an Intel® Pentium® 4 CPU 3.00 GHz processor and 2 GB of RAM. The operating system is SuSE Linux 9.1, because this is supported by all tools. Furthermore we ensured that the verification parameters and numerical solution methods of the tools match. For the numerical tools, e. g., the Jacobi method is used for solving systems of linear equations and the convergence accuracy ϵ is set to the default value 10^{-6}. For the statistical tools, we bound the probability of error (i. e. the chance of false negatives or positives) by $\alpha = \beta = 0.01$, which is the default setting for these tools, and half the width of the indifference region $\delta = 0.01$. The former agrees with possible choices of $\alpha = \beta$ from [145]. The choice of δ is somewhat arbitrary, and also taken from the literature.

Timing. In (probabilistic) model checking, two time factors are of interest: the *model construction time*, i. e. the time to build the internal representation from the input model, and the *model checking time*, i. e. the time to verify the property on the internal representation. We mainly focused on the bare model check time. One would often construct the model only once and then use it to verify multiple properties. In our comparison, we use the time as reported by the tools.

Memory usage. We measured the peak memory usage of the model checker, i. e. the amount of memory that is required for the verification problem at hand. More precisely, we recorded the virtual-memory size (RAM + swap) of the entire process (which includes model construction). We did so by running a script in parallel to the model checker that took a sample every 100 msec. Although this sampling method is not perfect, it gives us the means to conduct uniform measurements on all tools, and it provides a reasonable indication of the memory consumption of each tool. A disadvantage is that this method does not work for very small experiments that are too quick. Other methods, such as profiling tools, are less suitable as they e. g., require tool modifications.

Data collection. All experiments and measurement procedures were automated using shell scripts. This enabled us to easily repeat experiments many times and collect data in a uniform way. An experiment consists of verifying one property on one particular model using one of the model checkers. The tools are restarted before each experiment; this prevents the interference of e. g., caching on the measurements. Each experiment was repeated 20 times, except that experiments for which a single run took more than 30 minutes were repeated only three times. From the collected data, we calculated mean and standard deviation. The latter is determined using Student's t distribution, which takes the number of experiments into account. The maximal completion time for a single experiment was set to 24 hours, i. e., experiments that took

[3] A minor exception is E ⊢MC², where we added command line support to facilitate scripting.

longer were aborted. The verification time of these experiments is indicated in the results as ∞.

Model construction. The selected case studies were modeled using the model description language of each of the tools. For MRMC, E $\vdash$MC^2 and PRISM the models were readily available, viz., from the PRISM web page [115] or from the example models included in the tool distribution. Although the tools use different modeling languages, we require the models to be equivalent across all tools. Thanks to the export facility of PRISM version 3.0 beta, models in the PRISM language can be exported to the input format of E $\vdash$MC^2 and MRMC. The Ymer modeling language is almost identical to that of PRISM and only a few minor changes had to be made. The models for these four tools can thus safely be assumed to behave the same. The TQN and CPS case studies are provided in the standard distribution of the VESTA tool. Only for the BDP case study, a re-modeling effort was needed. We were not able to evaluate the SLE case study using VESTA due to parsing problems of the latter one.

We attempted to generate models as large as possible by varying the model parameters. In addition to the RAM size, two factors restrict the model size: the size of the *.tra* files used by MRMC and E $\vdash$MC^2 is limited to a maximum of 2 GB. In a few cases, we could not generate (and verify) our model as PRISM crashed due to a (known) problem of the CUDD package used for MTBDDs.

As MRMC and E $\vdash$MC^2 do not support a built-in modeling language, their overhead to generate a sparse matrix representation is low compared to the sparse matrix generation by PRISM. This aspect should be considered when interpreting the experimental results.

2.4.2 Experimental results and analysis

The experimental results are discussed per type of formula, allowing us to compare phenomena across the various case studies. The results are presented by histograms where the x-axis indicates the model parameters that determine the state space size, and the y-axis indicates the verification time (in seconds) or the memory consumption (in KB). Note that the y-axis is log-scale. The legend of the plots is given by Figure 2.5.

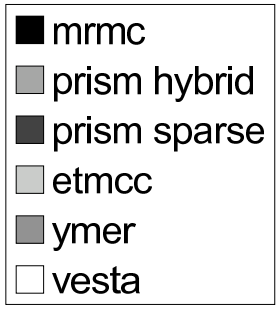


Figure 2.5: The legend

Almost sure reachability properties. We first consider unbounded-until formulas with probability bound ≥ 1. Figures 2.6 and 2.7 show the verification time and memory usage for the SLE case study for various (N, K) pairs. (Recall that N is the number of nodes, and K the identity range.) As PRISM checks qualitative properties in a symbolic manner regardless whether it uses the sparse or hybrid engine, there is neither a difference in runtime nor in memory consumption between PRISM^S and PRISM^H. On increasing model parameters, the memory consumption of MRMC grows gradually (as expected) whereas for PRISM^S and PRISM^H only a slight increase is observed. This is due to the fact that PRISM requires a large base memory for the JVM, the CUDD package (around 40 MB), and the

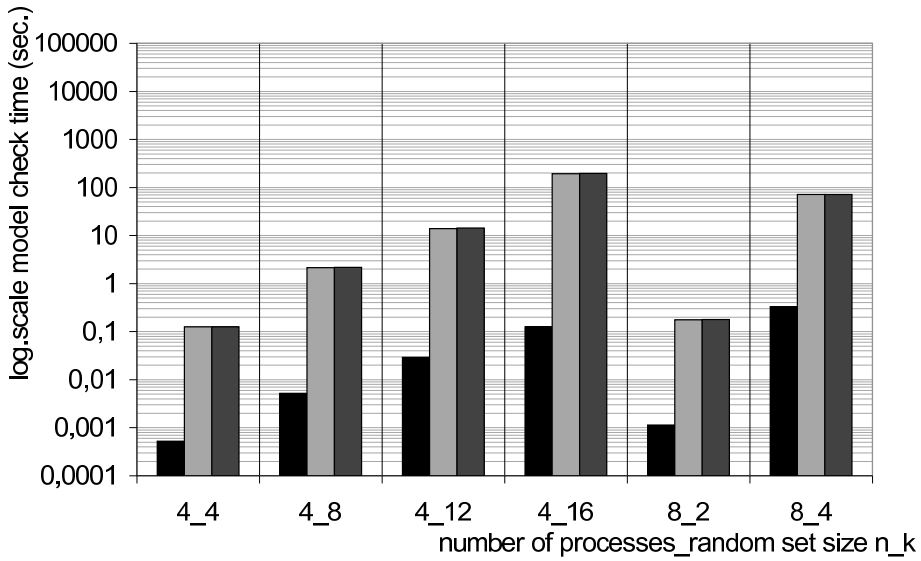


Figure 2.6: SLE, model check time: $\mathrm{P}_{\geq 1}\,(\Diamond\, elected)$

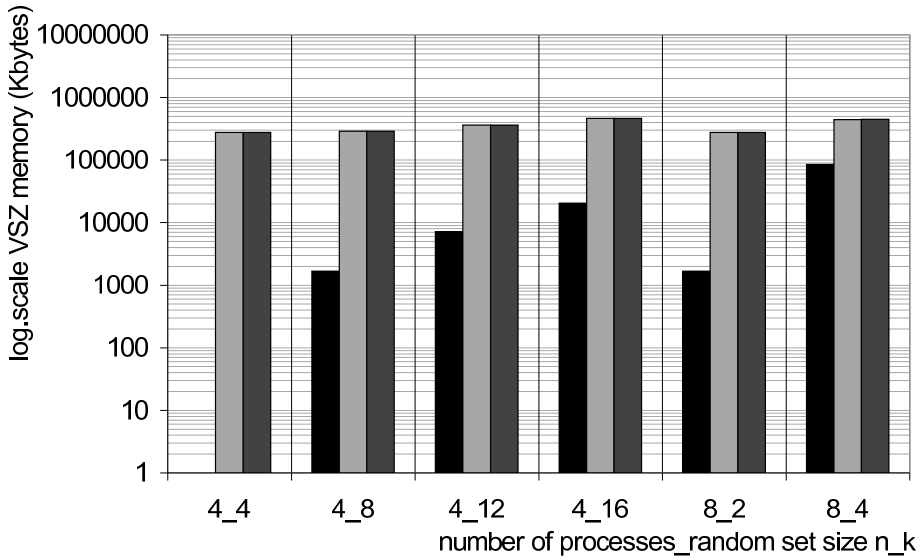


Figure 2.7: SLE, peak memory: $\mathrm{P}_{\geq 1}\,(\Diamond\, elected)$

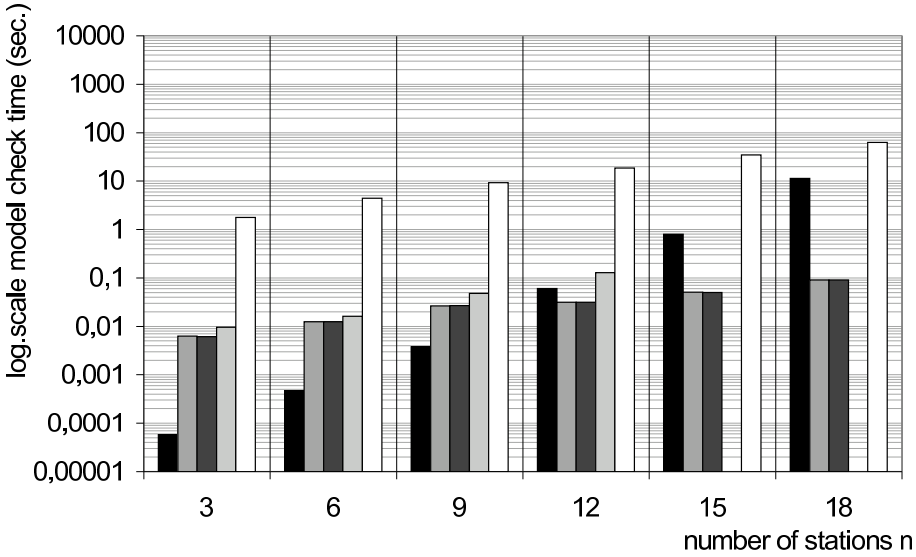


Figure 2.8: CPS, model check time: $busy_1 \implies \mathrm{P}_{\geq 1}(\Diamond poll_1)$

MTBDD it (always) generates. The MTBDD for this case study is not very compact, as indicated by the following table:

(N, K)	$(4,4)$	$(4,8)$	$(4,12)$	$(4,16)$	$(8,2)$	$(8,4)$
# MTBDD vertices	10K	1.6M	9M	27M	7K	10M
# states	0.8K	12K	62K	0.2M	2K	0.5M

As a result, PRISM needs substantially more memory than MRMC and the verification times differ up to several orders of magnitude. (For the smallest two problem instantiations, the memory consumption for MRMC is unavailable as its verification times are negligible.)

The SLE case study suggests that memory consumption for PRISM^S and PRISM^H is highly influenced by the MTBDD size. This observation is also substantiated by the CPS case study, for which the MTBDD sizes just increase slightly on a growth of the state space size:

N	3	6	9	12	15	18
# MTBDD vertices	112	367	765	1282	1942	2745
# states	36	0.6K	7K	74K	0.7M	7M

Observe that the MTBDD is very compact here, e. g., the model of 7 million states only requires 2745 MTBDD vertices, much less than in the SLE case study.

Some experimental results for a reachability property of the CPS case study are summarized in Figures 2.4.2 and 2.4.2. In contrast to the previous study, PRISM needs less memory than MRMC for large models due to the small MTBDD size. As before, there is no difference between PRISM^S and PRISM^H. For small models, MRMC is

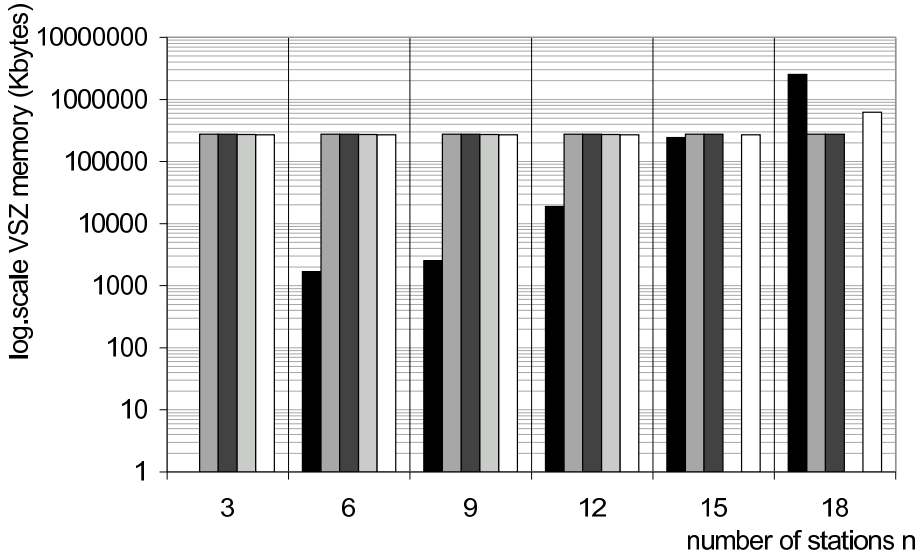


Figure 2.9: CPS, peak memory: $busy_1 \implies \mathrm{P}_{\geq 1}(\Diamond poll_1)$

faster and less memory intensive, but for $N \geq 15$, it is outperformed by PRISM^S. This effect is to be expected to be more drastic for larger values of N as PRISM^S is able to check the CPS for $N > 18$ (roughly 26 M states) rather efficiently. As the file size of the *.tra* file generated by PRISM for $N > 18$ exceeds 2 GB, we were unable to execute MRMC on it. For $N \geq 15$, E $\vdash$MC^2 runs out of memory. The performance of E $\vdash$MC^2 is worse than that of MRMC due to a less space-efficient sparse matrix representation, and the effect of the JVM. VESTA is about two orders of magnitude slower although—due to the use of Java—its memory usage is comparable to PRISM^S. The inefficiency of VESTA stems from the fact that it needs an excessive amount of sample paths to decide properties with bounds of the form ≥ 1, as shown in the following table:

N	3	6	9	12	15	18
# samples	34K	150K	395K	840K	1.6M	2.9M

Generally, statistical tools have difficulties to decide whether the probability of some property meets a bound if the actual probability and the bound are close. VESTA always gave the correct answer for these properties. For the BDP case study we experienced that for the property that almost surely eventually the population is maximal, VESTA reports an incorrect answer if the stopping probability—the likelihood that a sample path is stopped [122]—is not chosen appropriately. More precisely, if at some point during the simulation the stopping probability (in our case 0.05) is larger than that of reaching the state $N{=}M$ (in fact, a rare event), the sample path ends and it is concluded that $N{=}M$ is not reached. Re-simulation using a smaller stopping probability (e. g. 0.01) yields the correct answer. Note that Ymer is not used here as it does not support unbounded reachability properties.

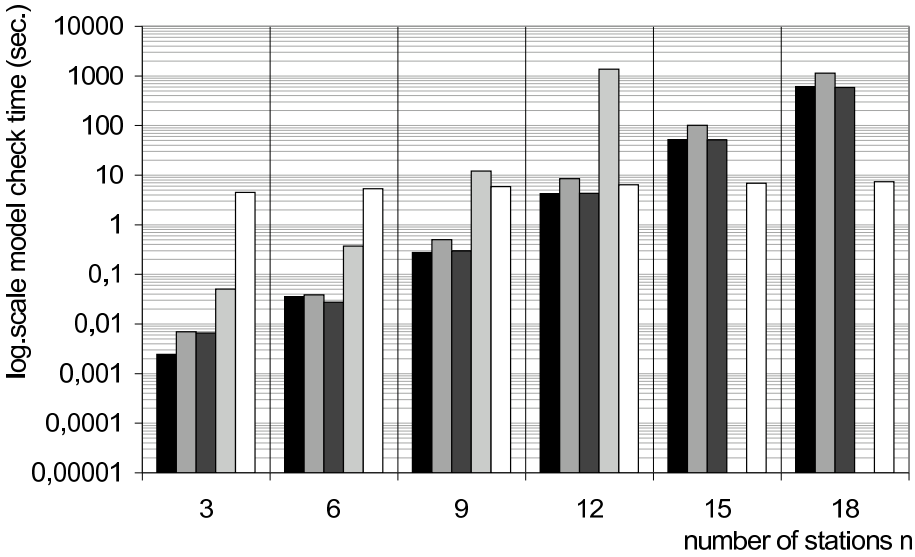


Figure 2.10: CPS, model check time: $busy_1 \implies \mathrm{P}_{\geq 0.5}\left(\Diamond^{[0,5]} poll_1\right)$

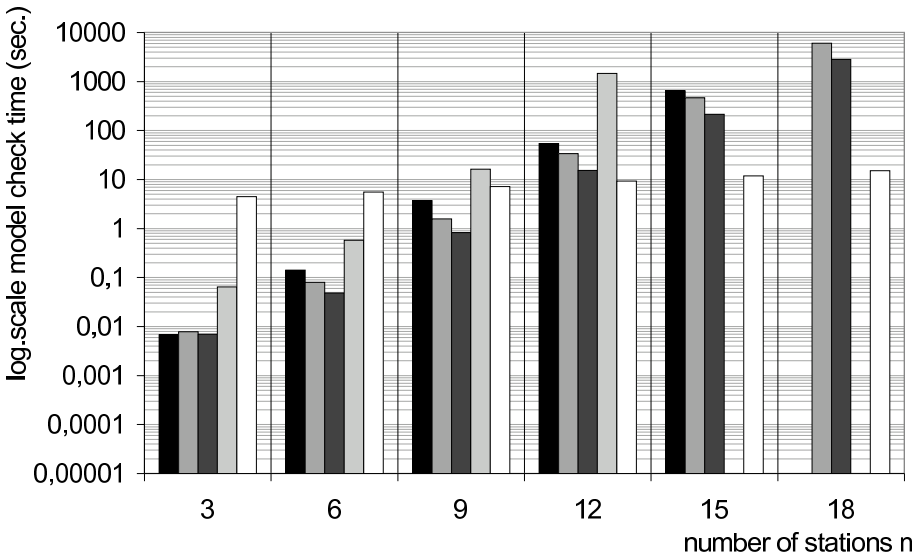


Figure 2.11: CPS, model check time: $busy_1 \implies \mathrm{P}_{\geq 0.5}\left(\Diamond^{[0,80]} poll_1\right)$

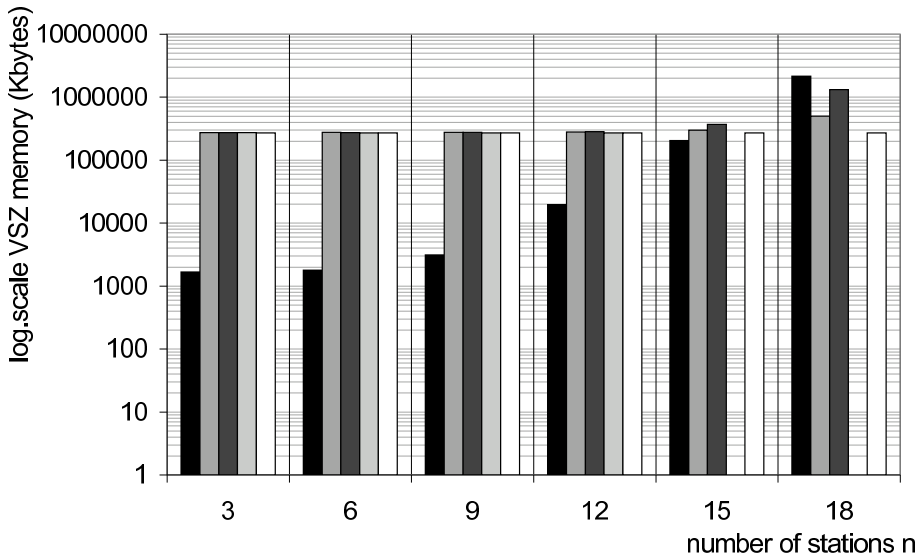


Figure 2.12: CPS, peak memory: $busy_1 \implies \mathrm{P}_{\geq 0.5}\left(\Diamond^{[0,t]} poll_1\right)$

Bounded-reachability properties. To show the effect of bounds, we consider a time-bounded variant of the property discussed before and observe what happens upon changing the time bound t. Figures 2.4.2 and 2.4.2 depict the verification times for the extreme bounds that we investigated in the CPS: t=5 and t=80, whereas Figure 2.4.2 depicts the memory consumption for arbitrary t—the memory consumption does not depend on t. The verification time required by MRMC is heavily influenced by t, e. g., for N=15 the verification time for t=20 is about four times longer than t=5. This is not surprising, as the time complexity of the underlying algorithm is linear in t. From t=30 on, the verification time is almost constant, due to a built-in steady-state detection, cf. Chapter 3. Besides, for t=80 and N=17, MRMC requires about 1700 seconds (not depicted), and we obtained a timer overflow for larger instantiations, i. e., the corresponding variable overflows. A similar behavior is obtained for E $\vdash$MC^2 but it runs out of memory rather quickly, as for simple reachability. PRISM^H is more efficient than PRISM^S due to the compact MTBDD (see previous case). As for MRMC, the verification time for PRISM^H and PRISM^S is linear in t, although this is less clear from the pictures due to the initial overhead of the MTBDD construction. A careful analysis of the log files reveals that the time *per iteration* is constant. Due to PRISM's steady-state detection, the verification time stops increasing around t=30. The verification time for VESTA for t=5 is rather constant as the number of samples (approx. 300, 000) is more or less the same for each N. For t=80 the number of samples slightly increases (it raises from 0.2M for N=3 to about 1.1M for N=18). This explains the small increase in run time in Figure 2.4.2. Unfortunately, VESTA gave wrong answers for low time bounds often: for $t = 5$, only 32.5% of the answers were correct. Note that the property has also been checked by Ymer, but as its run time is negligible—it immediately establishes that the initial state does not satisfy the premise of the implication—this is invisible in the figures. Ymer thus has an "excellent" performance,

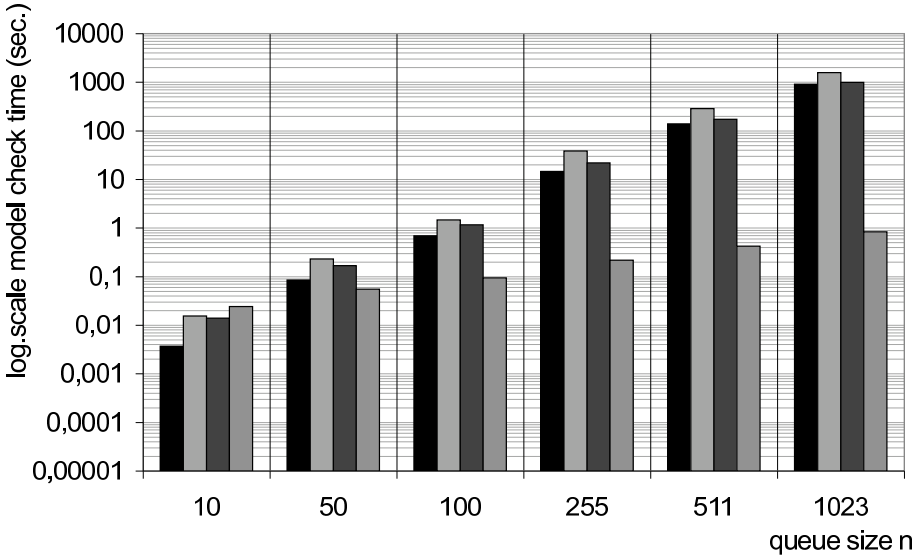


Figure 2.13: TQN, model check time: $\mathrm{P}_{\leq 0.01}\left(\Diamond^{[0.5,2]} \mathit{full}\right)$

but only checks the initial state whereas the other tools check *all* states. (VESTA also only provides answers for the initial state, but is unable to find the trivial satisfaction.)

Figures 2.4.2 and 2.4.2 show the results for checking a time-bounded property on the TQN case study. Ymer is for most cases much faster and smaller than all other tools. (For N=2 the verification time is too short to measure the memory consumption reliably.) As we have seen before, PRISM^H is more memory-efficient than PRISM^S, but the latter is faster. The memory usage of Ymer is less than VESTA, and for both simulation tools independent of the model size (as expected). As in the other case studies we see that due to the base memory overhead (JVM+CUDD) usage, the PRISM memory consumption is less dependent on the model size than MRMC, and E $\vdash$MC^2 is only able to handle relatively small models (up to few hundred thousands of states).

Figure 2.4.2 shows the timing for a bounded reachability property with both a positive lower and an upper bound (E $\vdash$MC^2 and VESTA do not support these bounds.) To check this formula, a model checker will calculate two reachability probabilities in different Markov chains and combine these results. The results are similar to the above, as expected: Ymer is, for most cases, the fastest tool; its runtime depends less on the model size than for the other tools. MRMC is slightly faster than PRISM^S, which is slightly faster than PRISM^H. The fact that Ymer is fast is also confirmed by checking such bounded property on the CPS case study, e. g. on N=16, Ymer just needs 1.2 sec whereas PRISM^S and MRMC require about 1500 sec, and PRISM^H about 3000 sec.

Steady-state properties. We only consider steady-state properties for CTMCs. The long-run operator for PCTL [4] is only supported by MRMC, and is therefore not used here. Ymer and VESTA do not support steady-state properties, basically as

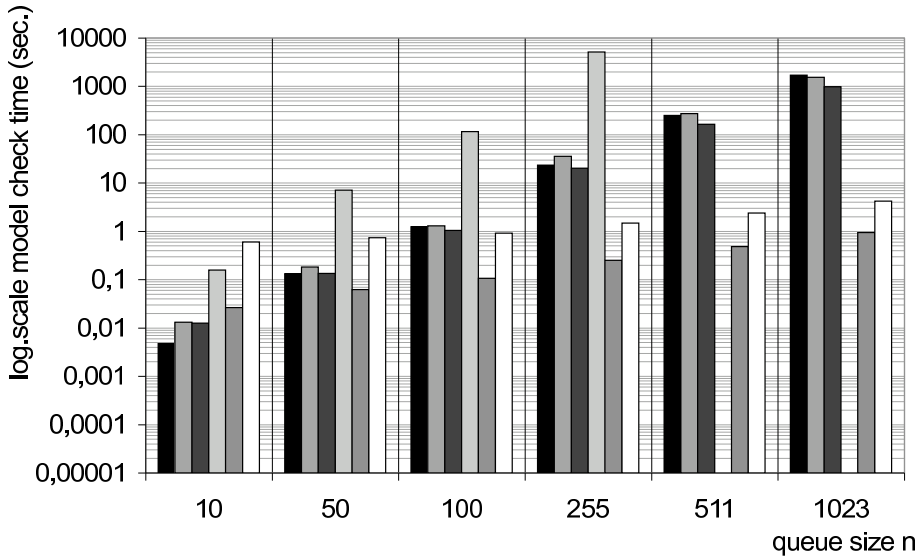


Figure 2.14: TQN, model check time: $\mathrm{P}_{\leq 0.01}\left(\Diamond^{[0,2]} \mathit{full}\right)$

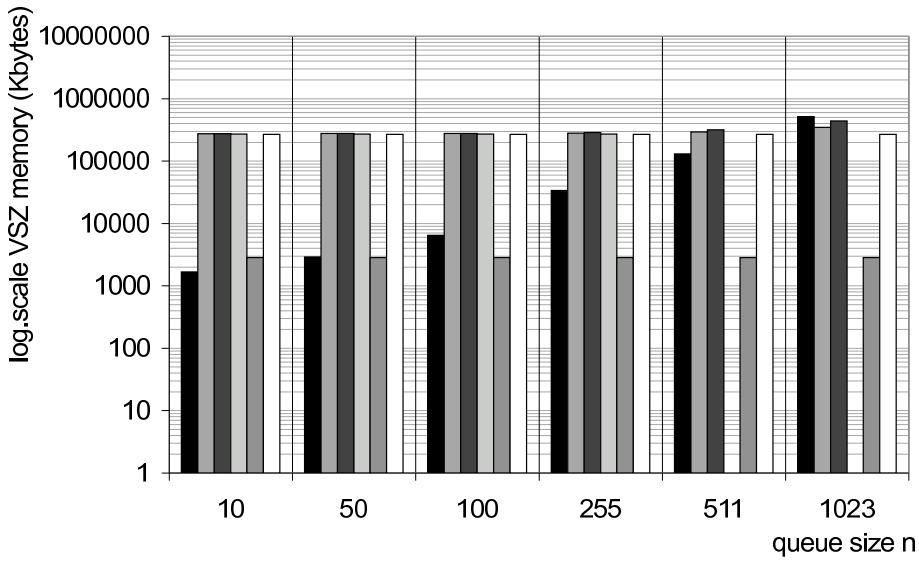


Figure 2.15: TQN, peak memory: $\mathrm{P}_{\leq 0.01}\left(\Diamond^{[0,2]} \mathit{full}\right)$

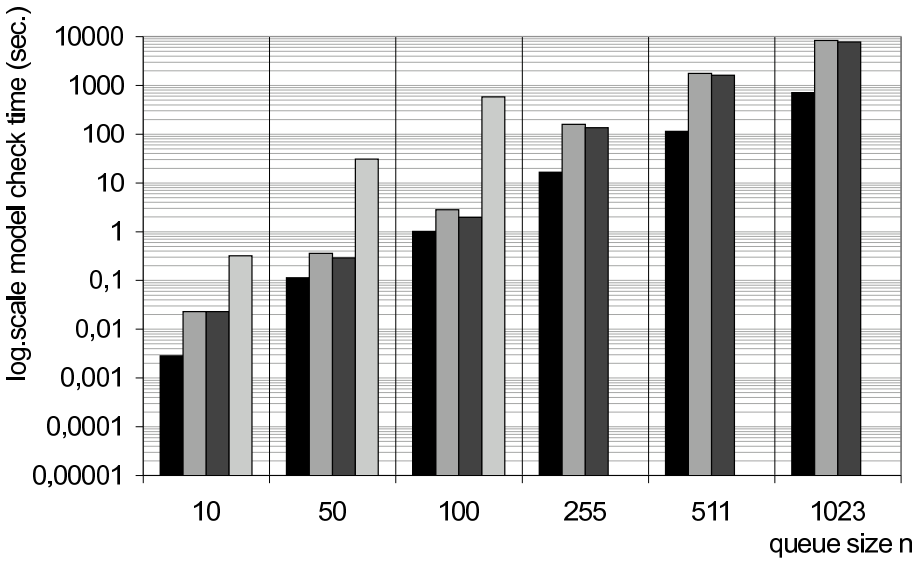


Figure 2.16: TQN, model check time: $S_{>0.2}\,(P_{>0.1}\,(X\ \mathit{snd}))$

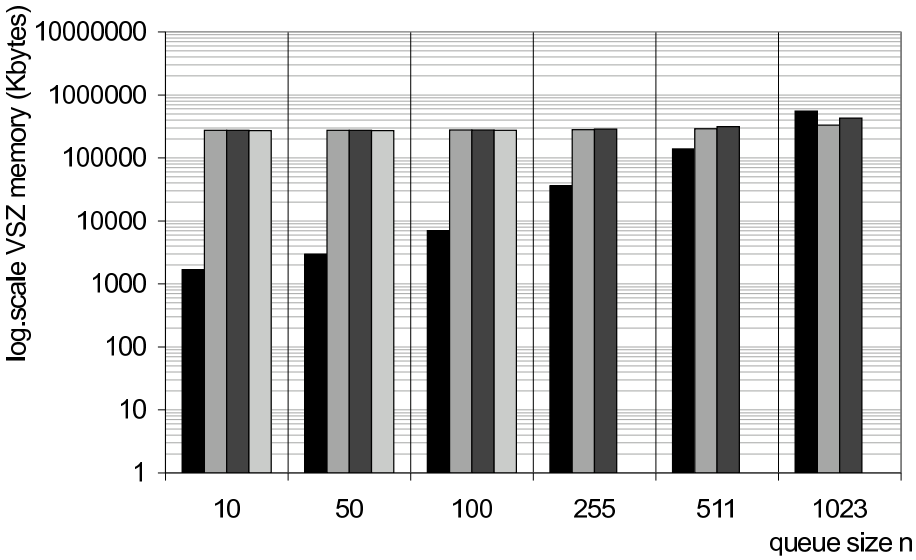


Figure 2.17: TQN, peak memory: $S_{>0.2}\,(P_{>0.1}\,(X\ \mathit{snd}))$

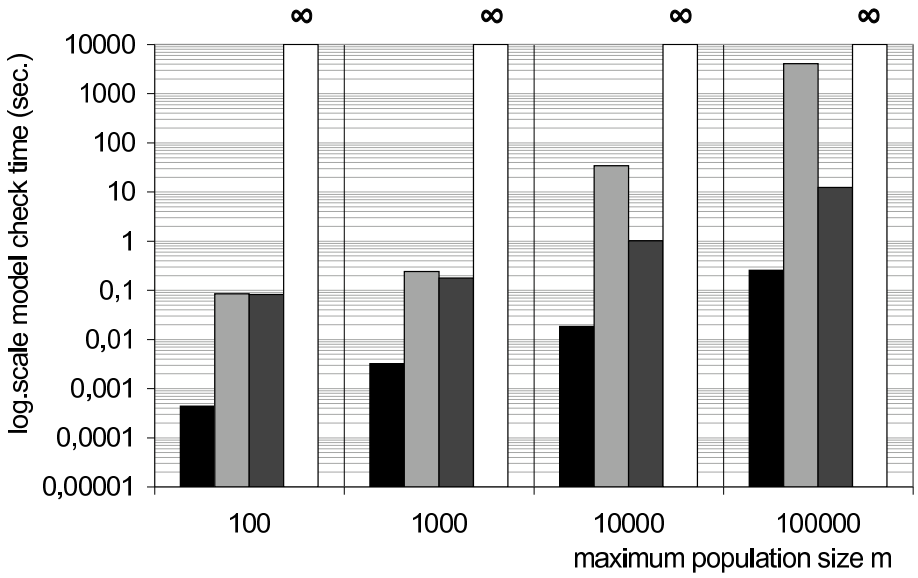


Figure 2.18: BDP, model check time: $\mathrm{P}_{\geq 0.8}\left(\mathrm{P}_{\geq 0.9}\left(\Diamond^{[0,100]}(N = 70)\right)\ \mathrm{U}\ (N = 50)\right)$

it is unclear on when to stop the sample path generation in their applied techniques. Figures 2.4.2 and 2.4.2 show the runtime and peak memory for a steady-state property in the TQN case study. The experiments show similar results as before. E $\vdash$MC^2 is the slowest tool and cannot handle large models (where $N > 100$). For the smaller models, the memory usage of PRISM is dominated by the overhead. For larger models, PRISM^S needs more memory than PRISM^H but is slightly faster. All experiments with steady-state formulas confirm our earlier observations: MRMC is faster and memory-wise more efficient than PRISM^S and PRISM^H, but for larger models, PRISM uses less memory than MRMC. The turn point, however, seems to occur at larger state spaces than experienced for the reachability properties.

Nested properties. We also checked the behavior on nested quantitative reachability properties. Figures 2.4.2 and 2.4.2 show the results of checking such property for the BDP case study. The tools check such nested formula in a bottom-up fashion, i. e., first the set of states satisfying the sub-formula is determined. The results are rather similar to the above findings. The MTBDD for the BDP case study is not very compact as the transition rates depend on the population size N, and as a result, most transition probabilities are distinct (resulting in many leaves in the MTBDD). As a result, MRMC outperforms PRISM^S and PRISM^H. Note however, that considered state spaces for this case study are relatively small which is favorable for MRMC. For all model instantiations, VESTA did not terminate simulation within 24 hours. We suggest as explanation that too many samples are required because the event N=70 is rather rare.

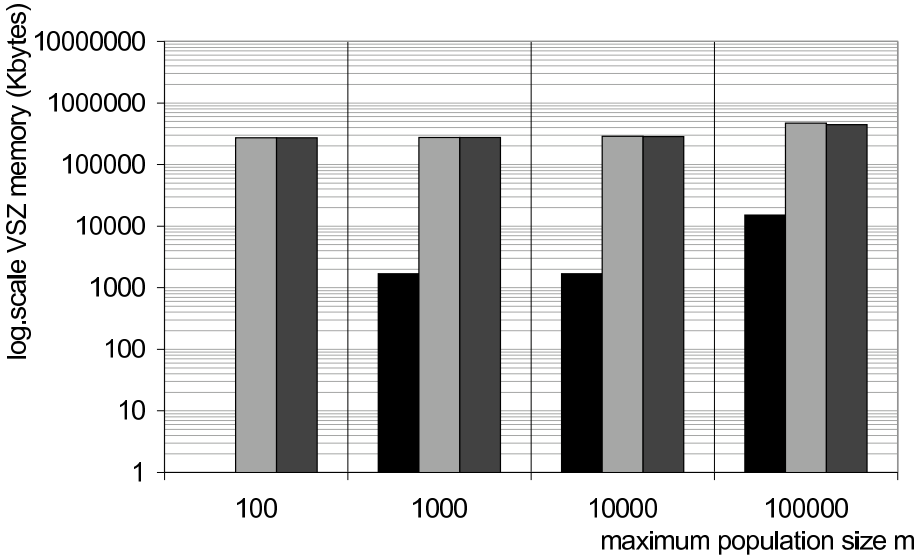


Figure 2.19: BDP, peak memory: $\mathrm{P}_{\geq 0.8}\left(\mathrm{P}_{\geq 0.9}\left(\Diamond^{[0,100]}(N = 70)\right)\ \mathrm{U}\ (N = 50)\right)$

2.4.3 Conclusion

We presented a performance comparison of five probabilistic model checkers. By ensuring that our experiments are repeatable, verifiable, statistically significant and free from external influences, our findings are based on a solid methodology.

From our experiments, we conclude that Ymer seems to be the fastest tool. Also, its memory usage is remarkably constant, hardly varying with the model size. Unfortunately, Ymer only supports bounded and interval until formulas. Also, as statistical tool, Ymer may report the wrong answer, and has done so during our experiments (in a few cases, as expected). In particular, Ymer outperforms the other statistical model checker VESTA: VESTA's memory consumption is also rather constant, but more in the order of PRISM's memory usage. However, its runtime varies a lot. For certain nested properties we checked, VESTA did not terminate within 24 h, even on a model with 100 states only.

As expected, PRISM^S is usually faster than PRISM^H at the cost of substantially greater memory usage. E $\vdash$MC^2 performs the worst in terms of memory, and frequently was unable to check models that were easy for the other tools.

For models up to a few million states, MRMC mostly performs better than PRISM^S both in time (although sparse matrix generation takes negligible time in MRMC compared to PRISM) and memory. This is mainly due to the overhead for MTBDD generation in PRISM. On larger models, PRISM^S and PRISM^H perform better. This effect is more apparent whenever the MTBDD representation is compact. As expected, PRISM^S is often faster than PRISM^H, but uses more memory. These results are summarized in Tables 2.4 and 2.5.

speed	E ⊢MC^2	MRMC	PRISM^S	PRISM^H	Ymer	VESTA
steady state	−	++	+	0/+ [a]	N/A	N/A
bounded until	−	+ [b]	+/++	0/+ [a]	++	+
unbounded until	−	+ [b]	+/++	+/++ [a]	N/A	−/0
nested	−	++	+	0/+ [a]	N/A [c]	−− [d]

[a]The time heavily depends on the MTBDD size.
[b]MRMC was faster in most cases, PRISM^S on larger models.
[c]The property contained operators not supported by Ymer.
[d]Based on one property, for which VESTA did not terminate.

Table 2.4: Speed performance comparison

Recommendations for MRMC Based on our experience, we can conclude that the considered version of MRMC (1.1.1b) performs well, comparing to other numerical model checking tools, such as E ⊢MC^2 and PRISM. The performance of MRMC, though, should be improved on larger models. This can be done in several ways: (1) by applying state-space reduction techniques, such as probabilistic bisimulation discussed in Chapter 4, (2) optimization of the implementation discussed in Section 2.5, and (3) improvement of the model checking algorithms, such as on-the-fly steady-state detection discussed in Chapter 3.

Comparison with the statistical model checking tools revealed that a statistical engine could be a fine complement to the numerical computations. The theoretical and experimental aspects of applying the discrete event simulations to probabilistic model checking are covered in Chapters 6 and 7.

memory	E ⊢MC^2	MRMC	PRISM^S	PRISM^H	Ymer	VESTA
steady state	−	+ [a]	+	+/++ [a] [b]	N/A	N/A
bounded until	−	+ [a]	+	+/++ [a] [b]	++	+ [c]
unbounded until	−	+ [a]	+/++	+/++ [a] [b]	N/A	0/+ [c]
nested	−	+ [a]	+	+/++ [a] [b]	N/A	N/A [d]

[a]MRMC used least memory in most cases. For larger models PRISM^S was between MRMC and PRISM^H, and PRISM^H was the best.
[b]The MTBDD size varied much with the case study.
[c]Fairly constant; inefficient for small models, efficient for large ones.
[d]Based on one property, for which VESTA did not terminate.

Table 2.5: Memory performance comparison

2.5 Implementation analysis

As it is mentioned in the previous section, MRMC, although rather fast and efficient, still needs some adjustments to be more competitive among the other model-checking tools. Therefore, to reveal the bottlenecks in our implementation, we focus on perfor-

mance profiling. Our analysis is based on the outputs provided by the tool *gprof*[4] that was applied to MRMC on the CPS case study (CSL properties). We do not profile the PCTL, PRCTL or CSRL model-checking procedures of MRMC because for PCTL the model-checking algorithms are simpler than for CSL, and the implementation for PRCTL and CSRL has not (yet) been optimized.

MRMC version The experiments listed below were performed on MRMC v1.2.1 (March 2006), as opposed to the experiments from Section 2.4 which were performed using MRMC v1.1.1b.

gprof In order to use *gprof* we first compiled MRMC with an extra GNU CC option: *'-pg'*. Then MRMC was run as usual on each of the tests, listed in the subsequent subsections. After each run an output file *"gmon.out"* was generated, containing the profiling data. For analyzing the gathered data the *gprof* was run, providing all the necessary statistics. *gprof* can produce several different output styles, we considered the *"flat profile"*, that shows the total amount of time MRMC spent executing each function. For more details on the *"flat profile"* read Appendix A.1.

Case study To clarify our choice of the representative case-study we must note that the steady-state and unbounded-reachability properties of CSL are checked in a similar manner as the long-run and unbounded-reachability properties of PCTL. Alternatively the time-bounded reachability properties of CSL are processed using more complex algorithms than that of PCTL. Therefore we could profile MRMC on either TQN or CPS case study. Among them we have selected CPS because for both case studies model checking times of steady-state and unbounded-reachability properties behave the same, but for time bounded-reachability problems there are more experiments showing that MRMC performed worse than PRISM on CPS than of TQN [110]. The properties used below, unless stated otherwise, are utilized and explained in Section 2.4.

For our experiments we considered the CPS model with $N = 17$. This is the largest value of N for which MRMC uses less than 2.0Gb of RAM. It is important because the machine we did experiments on has 2.0Gb RAM. Yet, the underlying model of CPS for $N = 17$ is still large enough to show the decrease of MRMC performance, when compared to PRISM.

Analysis In the following subsections we only consider three most time-consuming functions related to the model-checking algorithms of each property[5]. The profile data obtained with *gprof* is present in Appendix A.1.

2.5.1 Steady-state property

As in Section 2.4, we considered the property $\mathrm{S}_{<0.2}\,(\mathit{busy}_1 \wedge \neg \mathit{serve}_1)$, i. e., the steady-state probability that station 1 is waiting for the server is less than 0.2.

[4]This tool is a part of the GCC, the GNU Compiler Collection [35].

[5]The former is determined by the *% time* column of the *gprof* results and the latter by our knowledge of MRMC sources.

According to the obtained data the `multiplyUrowByConstAndAddToLUx` function takes 31.42% of time. This function is part of the Gauss-Seidel iterative method implementation and is responsible for multiplying a matrix row by a number and adding it to the next iteration vector. A more efficient implementation of this function can give a drastic improvement of model checking time.

Another function that is frequently called (about 70 million times) is `get_bit_val`. It gives access to a bit set element (see Section 2.2.1) and is already quite efficient but perhaps can be optimized further. As an alternative a different, and more efficient, data structure could be employed for storing sets of states.

The `set_mtx_val_ncolse` function calls, although time consuming, are less interesting. This function is invoked only for constructing the sparse matrix at the MRMC start-up and is irrelevant to model checking of the steady-state property. Therefore the third function we discuss is `getRoot`. It is used in the graph-traversal algorithm that searches for BSCCs and maps the MC states to their root values, see Section 2.2.2. At the moment, the implementation uses a simple array-based mapping. A more efficient hash table may improve the performance. Making the function *"inline"* can also improve the efficiency.

2.5.2 Reachability property

For model checking the property $busy_1 \implies \mathrm{P}_{\geq 1.0}(\Diamond poll_1)$, again the functions `get_bit_val` and `set_mtx_val_ncolse` show their importance. The first one takes 25.53% and the second 19.70% of the MRMC run time.

The next function is `get_exist_until`. It is used for transient analysis, see Section 2.1, and allows for determining the states from which the goal states may be reached via allowed states. The algorithm is based on a backward graph traversal [31] that is rather efficient. The implementation, though, may be improved by providing an easier sequential access to the bit-set elements.

The functions through `isWithinLineDelimiter` to `scan_number` are involved in loading the model and therefore are uninteresting. The function `get_always_until` is the last. As `get_exist_until`, it is widely used in transient analysis, and implements an algorithm described in [31]. The idea behind it is: finding the set of states from which the goal states are reached via allowed states with probability one. An improvement can be done on the level of manipulating the sets of states and memory management.

2.5.3 Bounded-reachability properties

Let us consider properties $busy_1 \implies \mathrm{P}_{\geq 0.5}\left(\Diamond^{[0,80]} poll_1\right)$ and $\mathrm{P}_{\geq 0.99}\left(\Diamond^{[40,80]} serve_1\right)$, i. e., the probability that station 1 will be served within the time bound $[40, 80]$ is at most 0.99.

Time-bounded reachability The function `multiply_mtx_cer_MV` is a serious bottleneck, since MRMC spent over 97% of run time in it. This function is used for multiplying a matrix by a vector where certain rows/vector elements are skipped, due to the states being made absorbing. The operation is vital for uniformization, see Section 2.1. A source code analysis revealed that it can be further optimized by at least simplifying the access to the sparse matrix elements.

The function `uniformization_plain` is the next candidate for a refinement. It implements the uniformization and is quite complex. The possible ways to improve its performance are to use a more efficient algorithm for computing Poisson probabilities (currently the Fox-Glynn algorithm is used) and/or to optimize the performance of matrix-vector multiplication.

The functions `set_mtx_val_ncolse`, `get_bit_val` and `get_exist_until` were already mentioned before.

Time-interval reachability The results obtained for interval-reachability property are very similar to the ones of the bounded-reachability property. The reason is that in the former case the uniformization has to be performed several times [8]. As before, functions `multiply_mtx_cer_MV`, `uniformization_plain`, `get_bit_val` and `get_exist_until` are the most time consuming.

2.5.4 Summary

The profiling results can be summarized as follows. First, the sparse-matrix representation should be further optimized with respect to numerical operations such as matrix-vector multiplication and operations specific to the numerical methods, e. g. Gauss-Seidel iterative method. Second, the bit-set structure, used to represent sets of states, has to allow for a faster access to its elements and more efficient set operations, such as union, intersection and complementation. Third, faster algorithms for computing Poisson probabilities and searching for BSCCs have to be utilized.

2.6 Implementation metrics

Implementation metrics, such as the number of *source-code lines*, *lines of comments*, *cyclomatic complexity* and the *development effort* estimates, are widely used to assess the quality and complexity of software as well as the implementation effort. In order to give a glimpse of the amount of work needed to implement a probabilistic model checker, we computed such metrics for MRMC v1.2.1 (January 2007) using the following tools:

- *Understand C/C++*: Version 1.4 (Build 402), evaluation version [75]
- *SLOCCount*: Version 2.26, released under GPL license [137]
- *CCCC*: Version 3.1.4, a Freeware product [96]

Understand C/C++ is a reverse engineering, documentation and metrics tool for C and C++ source code. It offers code navigation using a detailed cross reference, a syntax colorizing "smart" editor, and a variety of graphical reverse engineering views. Understand for C++ is an interactive development environment (IDE) designed to help maintain and understand large amounts of legacy or newly created C and C++ source code.

Understand, applied to the MRMC sources, produces the statistics in Table 2.6.

# Files	56
# Functions	364
# Lines	17013
# Lines blank	1503
# Lines of code [a]	6738
# Lines of comments [b]	8287
# Lines of comment per line of code	1.23

[a] Number of non-blank, non-comment lines of source code counted by the analyzer
[b] Number of lines of comment identified by the analyzer

Table 2.6: The MRMC metrics produced by Understand C/C++

SLOCCount is a suite of programs for counting physical source lines of code (*SLOC*) in potentially large software systems written in C, C++, Java, C# and many other languages. Originally, SLOCCount was developed by David A. Wheeler to count SLOC in a GNU/Linux distribution. SLOCCount, applied to the sources of MRMC, produces statistics in Table 2.7. Note that SLOCCount gives a development effort estimate

Lines of code [a]		
Language	# Lines	Percent
ANSI C	7210	94.32%
YACC	323	4.23%
LEX	111	1.45%
Total lines of code (SLOC)	7644	100%
Development effort estimate [b]	20.31 Person-Months	

[a] Totals grouped by language
[b] Using the basic COCOMO model [23, 20], Person-Months $= 2.4 \cdot \left(\frac{SLOC}{1000}\right)^{1.05}$

Table 2.7: The MRMC metrics produced by SLOCCount

which in fact is very close to the real effort spent on developing MRMC.

CCCC was developed as a testing ground for a number of ideas related to software metrics in an MSc project being undertaken by Tim Littlefair, under the supervision of Dr Thomas O'Neill at Edith Cowan University, Perth, Western Australia. CCCC, applied to the MRMC sources, produces statistics in Table 2.8. Note that the value for *Lines of code per line of comment* in Table 2.8 and the value for *Lines of comment per line of code* in Table 2.6 agree, since $\frac{1}{0.815} \approx 1.23$.

# Lines of code	6640
# Lines of comments	8146
McCabe's cyclomatic complexity [a]	1399
# Lines of code per line of comment	0.815
Cyclomatic Complexity per line of comment [b]	0.172
# Lines of code rejected by parser	339

[a] The number of linearly independent routes through a control flow graph of a program
[b] Indicates density of comments with respect to logical complexity of program

Table 2.8: The MRMC metrics produced by CCCC

The overall summary. The number of code lines reported by the considered tools varies slightly. The commercial tool Understand reports it to be 6738 whereas CCCC indicates it as 6640. The latter can be explained by the fact that CCCC can not correctly parse all the code and reports the number of lines rejected by parser to be 339, see Table 2.8. SLOCCount estimates the number of code lines as 7644 which we consider to be a better estimate since, unlike Understand and CCCC, it recognizes more source files, including those for yacc and lex. As a result we take the number of MRMC code lines to be around 7000. Both Understand and CCCC give close values for *Lines of comment per line of code*. The value 1.23 shows that MRMC sources are commented substantially. The *Development Effort Estimate*, given by SLOCCount (approximately 1.69 Person-Years), is close to what was spent on MRMC development. The Cyclomatic Complexity of MRMC, as reported by CCCC, is 1399. In the literature, values that exceed 50 are considered to indicate very complex and potentially highly unstable programs. Nevertheless, MRMC is stable and we think that it is a merit of the test suite, allowing for a fully automated testing of the tool. Therefore, the next section of this chapter is devoted to the MRMC test-suite and its metrics, such as the number of source-code lines and the test coverage.

2.7 MRMC test suite

The automated test-suite for MRMC allows to perform internal, functional and performance testing of the tool. The internal tests are targeted on testing the data structures, such as used for storing labels, the sparse matrix representation, etc. The functional tests assess the functionality of MRMC which includes its command-prompt parser, implemented model-checking algorithms, etc. Last but not least, the performance tests allow to evaluate the efficiency of implemented algorithms, such as probabilistic bisimulation minimization or on-the-fly steady-state detection. In addition to that, various memory-usage metrics of MRMC are collected. For more details on the test suite, consider reading its documentation.

The vast variety of used test cases includes various well-known case studies, discussed in Section 1.3: WGC, P2P, WC, CPS, RME, CP, SLE.

The test suite is freely distributed and can be obtained from [105]. Installation of

the test suite amounts to unpacking it into the MRMC-distribution folder, which as a result adds the *MRMC/test/* directory. The test suite is intended to be used on a Linux platform only and is not proven to work correctly under "Windows + Cygwin" or "Mac OS X".

2.7.1 The test-suite metrics

In order to estimate the effort spent on developing the test suite we have employed SLOCCount, the tool mentioned in Section 2.6. SLOCCount produces the statistics presented in Table 2.9. This statistics is incomplete but gives just a general idea, because many of our configuration scripts and files are not supported by SLOCCount. In addition, the total number of written lines (for most test-suite files), excluding data files for the test cases, is estimated to be about 8500 [6].

Lines of Code		
Language	# Lines	Percent
sh	832	56.45%
ANSI C	432	29.31%
awk	210	14.25%
Total lines of code (SLOC)	1474	100%
Development effort estimate	3.61 Person-Months	

Table 2.9: The test-suite metrics produced by SLOCCount

2.7.2 The test-suite coverage

In order to estimate the coverage, provided by the test suite, we have employed a test coverage program *gcov* [7]. Using it allowed us to find out how often and what lines of MRMC source code are actually executed during a run of the test suite. To use *gcov*, we first compiled MRMC with the special GNU CC options: *'-O0 -fprofile-arcs -ftest-coverage'*. Then the test suite was run in a regular fashion. As a result, for each source file of MRMC two files *.gcda* and *.gcno* were generated, containing the accumulated statistics. These files were analyzed with *gcov* and the summarized results are presented in the Table 2.10. As in Figure 2.2, we have divided MRMC sources into several logical components in order to simplify the representation. The coverage results only indicate the percentage of the source-code lines that was executed during the test runs. Therefore there is no information about the coverage of all alternative branches in the control-flow graph of the program. Nevertheless, we consider the given test coverage to be satisfying. For more details about the test coverage of MRMC see Appendix A.2.

[6] Using the standard Linux commands: `find and wc -l`

[7] Is a part of the GCC, the GNU Compiler Collection [35].

MRMC component	# Lines[a]	Coverage
Command-prompt interpreter[b]	654	61.16%
Input-file reader	175	81.14%
Options analyzer	233	72.53%
Runtime Settings	223	87.89%
PCTL model checking	123	100.00%
PRCTL model checking	186	96.77%
CSL model checking	345	94.49%
CSRL model checking	391	90.28%
Common model checking	479	91.65%
Internal Data Storage	806	83.13%
Bisimulation engine	558	90.32%
Numerical engines	362	78.18%
Total coverage of MRMC	4535[c]	83.46%

[a]The total number of code lines.

[b]We give coverage for the actual parser files generated by Yacc and Lex: *lex.yy.c*, *y.tab.c*, plus the source file *parser_to_core.c*.

[c]The total number of source lines for MRMC, according to *gcov* is smaller than reported by other tools (see Section 2.6). The reason is that *gcov* counts only the meaningful lines, for example it omits lines that have only closing braces on them.

Table 2.10: The test-suite coverage, produced by *gcov*

2.8 MRMC and the third-party projects

In this section we present third-party projects that use MRMC and also model checking tools that interface with it.

GreatSPN and MRMC GreatSPN v2.0 [99] is a software package being developed by the Department information at the Università di Torino, Italy in cooperation with the Dipartimento di Informatica at the Università del Piemonte Orientale, Alessandria, Italy.

The package is used for modeling, validation, and performance evaluation of distributed systems using Generalized Stochastic Petri Nets and their colored extension: Stochastic Well-formed Nets. It provides a friendly framework to experiment with timed Petri net based modeling techniques, and implements efficient analysis algorithms to allow its use on rather complex applications. GreatSPN uses MRMC as a backend for CSL model checking [28].

Prism and MRMC PRISM [68] is a probabilistic model checker being developed by the School of Computer Science at the University of Birmingham, United Kingdom.

The tool supports three kinds of models: DTMC, CTMC and MDP (Markov Decision Processes). System models are described using the PRISM modeling language which is a high-level state-based description language, based on the Reactive Modules formalism of Alur and Henzinger [3]. The PRISM property specification language incorporates PCTL and CSL logics, as well as extensions for quantitative specifications and costs/rewards. PRISM (since version 3.0) is capable of exporting its models [107] into the MRMC input-file formats.

Performance Evaluation Process Algebra (PEPA) Performance Evaluation Process Algebra (PEPA) [67] is an algebraic process-oriented language for modeling concurrent systems. The process algebra is being mainly developed in Laboratory for Foundations of Computer Science, University of Edinburgh, United Kingdom.

Performance of a PEPA model can be evaluated by deriving and analyzing the underlying CTMC. PEPA modelers are provided with the PEPA Workbench [135], an Eclipse-platform [49] application for managing the models. One of the PEPA Workbench features is an Eclipse wizard for exporting PEPA models into the MRMC input-file formats.

Heuristics-Guided Dependability Analysis The project is carried out at the chair of Software Engineering at the Universität des Konstanz, Germany.

It is aimed at generating diagnostics information for stochastic models [2], also known as failure traces or counterexamples. The main concern is with application of heuristic-guided search algorithms, to efficiently determine diagnostic traces which carry large amount of probability. Such traces are an essential tool for diagnostics and debugging of stochastic models. In the course of the work on this topic, a prototype tool called DiPro has been implemented and linked to PRISM. Among the other goals of the project is an implementation of interfaces to MRMC.

Reachability analysis in continuous-time Markov decision processes The project is carried out in the Dependable Systems & Software Group at the Universität des Saarlandes, Germany.

One of the project objectives is to link an industrial state-of-the-art modeling tool (STATEMATE) to academic state-of-the-art analysis algorithms [19]. STATEMATE models are transformed into uniform continuous-time Markov decision processes (uCTMDPs) which are used for analyzing the timed-reachability properties [12] with the help of the MRMC tool extension.

The tool chain (STATEMATE – extended MRMC) had been complete and was applied to a set of well-known case studies [79], e.g. the European Train Control System (ETCS) [19], fault-tolerant workstation cluster [58, 62], and mutual exclusion problem [52]. Currently the timed-reachability algorithms for uCTMDPs are being integrated into the new release of MRMC v1.3.

2.9 Conclusion

In this chapter we have presented the probabilistic model checker named MRMC that is being developed at the Formal Methods and Tools group, University of Twente, The Netherlands and Software Modeling and Verification group, RWTH-Aachen, Germany.

We first introduced the tool in Section 2.1 by means of several simple examples and then in Section 2.2 discussed the tool internals, such as the implemented algorithms, the architecture and the underlying data structures. This helped us to analyze the results of comparative experimental study of MRMC and the probabilistic model checkers E $\vdash$MC^2, PRISM, YMER and VESTA provided in Section 2.4. The main conclusion of this analysis was that MRMC v1.1.1b is highly competitive with other tools, especially when applied to small and medium size models (up to several millions states). Nevertheless, in order to analyze the possible bottlenecks of the implementation, in Section 2.5 we carried out the performance profiling of MRMC v1.2.1 (the latest available version at the time of profiling) and gave detailed recommendations for the tool improvements, which were partially realized in MRMC v1.2.2 released in August 2007. To complete the overview of the tool and its development we presented the various source-code metrics of MRMC in Section 2.6 and the MRMC test suite in Section 2.7. Finally, in Section 2.8 we discussed the applications of MRMC in several third-party projects aimed at: the validation and performance evaluation of Stochastic Well-formed Nets, counter-examples generation, and model checking CTMDPs.

Chapter 3

On-The-Fly Steady-State Detection

When performing transient analysis for a CTMC, see Section 1.1.2, it is common practice—in particular in case of large time spans—to use a built-in steady-state detection technique [97, 145]. The underlying idea is to be able to detect whether the CTMC has reached an equilibrium before the end of the (large) time bound. Whenever such equilibrium is detected, the transient computation can be stopped thus saving expensive computational steps. The criteria for detecting such equilibria when guaranteeing a given overall inaccuracy are, however, not always clear and may lead to the detection of premature equilibria. This may happen, for instance, when the probability mass in the CTMC under consideration only changes slightly in a series of computational steps due to a "slow" movement.

As checking time-bounded reachability properties of CSL reduces to transient analysis, cf. Section 1.2.2, on-the-fly steady-state detection can be exploited in probabilistic model checking. Numerical probabilistic model checkers such as PRISM, $\mathrm{E} \vdash \mathrm{MC}^2$, discussed in Section 1.4, and their variants for stochastic Petri nets (such as GreatSPN [38] and the APNN Toolbox [25]) have adopted this technique for model checking CSL *as it is*, without tailoring it to the specific nature of time-bounded reachability. Thus they can not avoid detecting a premature stationarity, sometimes providing unreliable model-checking results.

Further, we present a detailed analysis of the use of on-the-fly steady-state detection in this setting. Since computing Poisson probabilities, an essential ingredient in CTMC transient analysis, is typically done using the procedure suggested by B. L. Fox and P. W. Glynn [50], we start by revisiting and slightly sharpening their results. Based on this, we improve the known criteria to decide whether an equilibrium has been reached for on-the-fly steady-state detection in case of CTMC transient analysis [97] and CSL model checking [143, 145]. Furthermore, for the latter a simple procedure is proposed to safely detect equilibria. This is done by exploiting the structure of the CTMC that is obtained when reducing time-bounded reachability to transient analysis. Note that this algorithm is correct in the sense that premature stationarity is *never* detected.

Experimental results complete the discussion and show the impact of our theoretical achievements. By means of an artificial, though extremely simple CTMC, we show that

various existing probabilistic model checkers detect a premature equilibrium resulting in incorrect verification results. We report similar observations for several case studies discussed in Section 1.3, namely: the workstation cluster and the IEEE 802.11 group communication protocol. The former one is an example that has established itself as a benchmark problem for probabilistic model checking. Our results for the latter one confirm the premature steady-state detection phenomenon reported in a recent analysis of the protocol in [100]. These experiments clearly indicate the benefits of our algorithm. Based on these observations, we firmly believe that the presented results improve current probabilistic model-checking technology.

The remainder of this chapter is organized as follows: Section 3.1 provides an introduction of the steady-state detection problem both in application to transient analysis and model checking of CTMCs. Section 3.2 presents the slight refinement of the Fox-Glynn error-bound. Sections 3.3 and 3.4 contain the main contribution of this work; these sections present new criteria for detecting equilibria during time-bounded reachability, and the algorithm to safely detect steady state. Section 3.5 reports on the conducted experiments. The time-complexity and some results on empirical evaluation of the suggested steady-state detection procedure are presented in Sections 3.6. The conclusions are given in Section 3.7.

Most results provided in this chapter are published as [86] and [82].

3.1 Introduction

In this section we first recall several facts about computing the transient and time-bounded reachability probabilities for a CTMC $(S, \mathbf{Q}, L)$, for more details consider reading Sections 1.1.2 and 1.2.2. Then we discuss the known on-the-fly steady-state detection techniques and their application to computing the before-mentioned probabilities.

The vector of transient probabilities $\overrightarrow{\pi^{o,*}(t)}$ for the CTMC at time t, when given the initial distribution $\overrightarrow{p^o(0)}$, is computed as:

$$\overrightarrow{\pi^{o,*}(t)} = \sum_{i=0}^{\infty} \gamma_i(t) \cdot \overrightarrow{p^o(i)}, \tag{3.1}$$

where $\gamma_i(t)$ is the Poisson density function, for all $i \geq 1$ we have $\overrightarrow{p^o(i)} = \overrightarrow{p^o(i-1)} \cdot \mathbf{P}$, and $\mathbf{P}$ is the uniformized CTMC.

As it is mentioned earlier, the time-bounded reachability problem of CSL logic can be reduced to transient analysis. In case of *local model checking* (forward-reachability, Algorithm 1), the analysis needs to be carried out for a single state $s \in S$ only, which corresponds to the computation of transient probability $Prob\left(s,\ \mathcal{A}\ \mathrm{U}^{[0,t]}\ \mathcal{G}\right) = \sum_{j \in \mathcal{G}} \pi^{o,*}_j(t)$, by employing Equation (3.1) with $\overrightarrow{p^o(0)} = \overrightarrow{1_{\{s\}}}$.

For *global model checking* (backward-reachability, Algorithm 2), the validity of a logical property needs to be checked in every state and thus the probabilities must be computed for all initial states. The latter can be done by computing the vector:

$$\overrightarrow{\pi^{*}(t)} = \sum_{i=0}^{\infty} \gamma_i(t) \overrightarrow{p(i)}, \tag{3.2}$$

with $\overrightarrow{p\,(0)} = \overrightarrow{1_{\mathcal{G}}}$, and $\overrightarrow{p\,(i)} = \mathbf{P} \cdot \overrightarrow{p\,(i-1)}$ for all $i \geq 1$. Then for all $s \in S$ we have $Prob\left(s,\ \mathcal{A}\ \mathrm{U}^{[0,t]}\ \mathcal{G}\right) = \pi^*_s\,(t)$.

For both Equations (3.1) and (3.2) the infinite sum is normally computed using the Fox-Glynn algorithm, cf. Section 1.1.2. Another important observation is that, because $\overrightarrow{p^o\,(i)}$ and $\overrightarrow{p\,(i)}$ are iteration-step vectors for the Power method, cf. Section 1.1.1, the steady-state detection can be employed, allowing the increase of the computation efficiency. In the sequel we discuss the known steady-state detection techniques in application to computing probabilities $\overrightarrow{\pi^{o,*}\,(t)}$ and $\overrightarrow{\pi^*\,(t)}$. We also point out the flaws and limitations of these approaches.

3.1.1 Transient probabilities

Malhotra *et. al.* [97] present a numerical method, which takes into account steady-state detection, for computing CTMC transient probabilities, see Equation (3.1), with an overall error bound ε. For the sake of this dissertation, we state their result in the following form:

Claim 1 *[97] Let $(S,\ \mathbf{P},\ L)$ be an aperiodic DTMC with initial distribution $\overrightarrow{p^o}$ and steady-state distribution $\overrightarrow{p^{o,*}}$. If for some K and $\delta > 0$ it holds that $\forall i \geq K$: $\left\|\overrightarrow{p^{o,*}} - \overrightarrow{p^o\,(i)}\right\|_v \leq \delta$, where $\|.\|_v$ is an arbitrary vector norm, then for*

$$\overrightarrow{\pi^{o,*}\,(t)} = \sum_{i=0}^{\infty} \gamma_i(t) \cdot \overrightarrow{p^o\,(i)}$$

and for inaccuracy $\varepsilon > 0$:

$$\overrightarrow{\pi^o\,(t)} = \begin{cases} \overrightarrow{p^o\,(K)} & , \text{ if } K < \mathcal{L}_\epsilon \\ \sum_{i=\mathcal{L}_\epsilon}^{K} \gamma_i(t)\overrightarrow{p^o\,(i)} + \overrightarrow{p^o\,(K)}\left(1 - \sum_{i=0}^{K} \gamma_i(t)\right) & , \text{ if } \mathcal{L}_\epsilon \leq K \leq \mathcal{R}_\epsilon \\ \sum_{i=\mathcal{L}_\epsilon}^{\mathcal{R}_\epsilon} \gamma_i(t)\overrightarrow{p^o\,(i)} & , \text{ if } K > \mathcal{R}_\epsilon \end{cases} \tag{3.3}$$

the following inequality holds:

$$\left\|\overrightarrow{\pi^{o,*}\,(t)} - \overrightarrow{\pi^o\,(t)}\right\|_v \ \leq\ 2\delta + \frac{\varepsilon}{2}$$

Here, $\mathcal{L}_\epsilon$ and $\mathcal{R}_\epsilon$ are computed using the Fox-Glynn algorithm (see below), such that $\sum_{i=0}^{\mathcal{L}_\epsilon - 1} \gamma_i(t) \leq \frac{\varepsilon}{2}$, and $\sum_{i=\mathcal{R}_\epsilon+1}^{\infty} \gamma_i(t) \leq \frac{\varepsilon}{2}$.

Claim 1 can now be used to obtain a criterion for guaranteeing an overall inaccuracy of $\varepsilon > 0$ for transient analysis with on-the-fly steady-state detection.

Corollary 3 *Under the same conditions as Claim 1:*

$$\left\|\overrightarrow{p^{o,*}} - \overrightarrow{p^o\,(K)}\right\|_v \leq \frac{\varepsilon}{4} \quad \textit{implies} \quad \left\|\overrightarrow{\pi^{o,*}\,(t)} - \overrightarrow{\pi^o\,(t)}\right\|_v \leq \varepsilon \tag{3.4}$$

As $\left\|\overrightarrow{p^{o,*}} - \overrightarrow{p^o\,(K)}\right\|_v$ is not known during computations (since $\overrightarrow{p^{o,*}}$ is unknown, and typically not computed a priori as this is computationally too expensive), [97] suggests to use the absolute convergence test, i.e. to replace the premise in Equation (3.4) by:

$$\left\|\overrightarrow{p^o\,(i)} - \overrightarrow{p^o\,(i{+}M)}\right\|_v \leq \frac{\varepsilon}{4} \quad \text{for } M > 0$$

Accordingly, $\overrightarrow{p^o\,(K)}$ with $K = i{+}M$ is used as an approximation of the real steady-state distribution. This approach thus boils down to comparing probability vectors that are M iterations apart. Once these probability vectors are close enough, it is assumed that the CTMC has reached an equilibrium. This approach, of course, has the drawback that due to the use of an approximation of the stationary probability, an equilibrium may be detected prematurely. A detailed analysis revealed that in deriving the above result in [97], an important ingredient of the Fox-Glynn algorithm is not considered, viz. the so-called weights (cf. Section 1.1.2). It will be shown in the remainder of this chapter that weights play an important role to strengthen the premise of (3.4), and thus to obtain a *safer* criterion to detect equilibria. Also, it should be noted that Claim 1 does not hold for an arbitrary norm $\|.\|_v$. In fact, the additional condition $\|\overrightarrow{p}\|_v \leq 1$ for any distribution vector $\overrightarrow{p}$, is required.

3.1.2 Time-bounded reachability

The steady-state detection for transient analysis of CTMCs discussed in Section 3.1.1 is applicable to the forward computations, see Equation (3.2), in a straightforward way. Steady-state detection for backward computations has been recently discussed in [145]. The approach by Younes *et al.* is based on the following result.

Claim 2 *Let $(S,\ \mathbf{P},\ L)$ be an aperiodic DTMC with $Ind \subseteq S$ such that $\forall j \in Ind$: $P(j,j) = 1$, $\overrightarrow{p\,(i)} = \mathbf{P}^i \cdot \overrightarrow{1_{Ind}}$ and steady-state vector $\overrightarrow{p^*}$. If for some K and $\delta > 0$ it holds that $\forall i \geq K : \left\|\overrightarrow{p^*} - \overrightarrow{p\,(i)}\right\|_v \leq \delta$, where $\|.\|_v$ is an arbitrary vector norm, then for*

$$\overrightarrow{\pi^*\,(t)} = \sum_{i=0}^{\infty} \gamma_i(t)\overrightarrow{p\,(i)}$$

and for inaccuracy $\varepsilon > 0$:

$$\overrightarrow{\pi\,(t)} = \begin{cases} \overrightarrow{p\,(K)} & , \text{ if } K < \mathcal{L}_\epsilon \\ \sum_{i=\mathcal{L}_\epsilon}^{K} \gamma_i(t)\overrightarrow{p\,(i)} + \overrightarrow{p\,(K)}\left(1 - \sum_{i=\mathcal{L}_\epsilon}^{K} \gamma_i(t)\right) & , \text{ if } \mathcal{L}_\epsilon \leq K \leq \mathcal{R}_\epsilon \\ \sum_{i=\mathcal{L}_\epsilon}^{\mathcal{R}_\epsilon} \gamma_i(t)\overrightarrow{p\,(i)} & , \text{ if } K > \mathcal{R}_\epsilon \end{cases} \tag{3.5}$$

the following inequality holds:

$$\left\|\overrightarrow{\pi^*\,(t)} - \overrightarrow{\pi\,(t)}\right\|_v \leq 2\delta + \frac{\varepsilon}{2}$$

Here $\mathcal{L}_\epsilon$, and $\mathcal{R}_\epsilon$ are computed using the Fox-Glynn algorithm, such that $\sum_{i=0}^{\mathcal{L}_\epsilon - 1} \gamma_i(t) \leq \frac{\varepsilon}{2}$ and $\sum_{i=\mathcal{R}_\epsilon+1}^{\infty} \gamma_i(t) \leq \frac{\varepsilon}{2}$.

In [145], this result has led to the following practical check for steady-state:

$$\left\|\overrightarrow{p^*} - \overrightarrow{p\,(K)}\right\|_v \leq \frac{\varepsilon}{8} \quad \text{implies} \quad \forall j \in S : -\frac{\varepsilon}{4} \leq \pi_j^*\,(t) - \pi_j\,(t) \leq \frac{3}{4}\varepsilon \tag{3.6}$$

As before, since $\overrightarrow{p^*}$ is not known during computations, the absolute convergence test is used instead. That is, the premise is replaced by $\left\|\overrightarrow{p\,(i)} - \overrightarrow{p\,(i+M)}\right\|_v \leq \frac{\varepsilon}{8}$. The vector $\overrightarrow{p\,(K)}$ with $K = i+M$ is thus used as an approximation of the steady-state vector. Whereas for the forward analysis case, the convergence test bound equals $\frac{\varepsilon}{4}$ (cf. Equation (3.4)), for the backward analysis this is $\frac{\varepsilon}{8}$ (cf. Equation (3.6)). One may question how safe (and tight) this criterion for equilibrium detection is. As these results are based on [97], the drawbacks of this method are inherited. A detailed look at the Equations (3.3) and (3.5) for the case $\mathcal{L}_\epsilon \leq K \leq \mathcal{R}_\epsilon$ reveals that the second summation for the backward case starts at $i = \mathcal{L}_\epsilon$ rather than $i = 0$. The justification for this change is unclear, but has a non-negligible impact on the bound. To be more precise, this change of the summation index implicitly increases the error bound and this is not taken care of. More importantly, though, the analysis resulting in Claim 2 is based on the assumption that the steady-state detection error is two-sided, whereas—due to the backward nature of the algorithm— it is in fact *one-sided*.

3.2 Fox-Glynn error bound revisited

In Sections 1.1.2 and 1.2.2 we discuss the application of the Fox-Glynn algorithm to computing the transient probabilities of CTMCs and model checking of the time-bounded reachability property (CSL logic). This algorithm allows to compute truncation points $\mathcal{L}_\epsilon$ and $\mathcal{R}_\epsilon$ with the weights W and $w_i(t)$ for $\mathcal{L}_\epsilon \leq i \leq \mathcal{R}_\epsilon$ in such a way that for a real-valued function f we have:

$$\left|\sum_{i=0}^{\infty} \gamma_i(t) f(i) - \frac{1}{W} \sum_{i=\mathcal{L}_\epsilon}^{\mathcal{R}_\epsilon} w_i(t) f(i)\right| \leq \varepsilon \cdot \|f\|,$$

see Proposition 2 for the exact formulation. In other words the inequality above provides the error bound for the approximation of the infinite sum $\sum_{i=0}^{\infty} \gamma_i(t) f(i)$. The following refinement of this error bound can be made for the case when f does not change sign, i.e., $f(i) \leq 0$ or $f(i) \geq 0$, for all i. Which is especially important because, as we will see in the next section, it allows to refine the error bounds for the transient and time-bounded reachability probabilities computed using the steady-state detection technique.

Proposition 4 *For real-valued function f* that does not change sign*, and a Poisson density function $\gamma_i(t)$, if $\sum_{i=\mathcal{L}_\epsilon}^{\mathcal{R}_\epsilon} \gamma_i(t) \geq 1 - \frac{\varepsilon}{2}$ then the following holds:*

$$\left|\sum_{i=0}^{\infty} \gamma_i(t) f(i) - \frac{1}{W} \sum_{i=\mathcal{L}_\epsilon}^{\mathcal{R}_\epsilon} w_i(t) f(i)\right| \leq \frac{\varepsilon}{2} \cdot \|f\|.$$

Proof See the proof of Proposition 35 from Appendix B.1. □

3.3 Improved steady-state detection

In this section, we provide new error bounds for on-the-fly steady-state detection during transient analysis and time-bounded reachability. Detailed proofs are provided to substantiate our claims.

Remark. The error estimate in [97] is norm based and relies on the *geometrical convergence* of power iterations for an aperiodic DTMC. The geometrical convergence is usually proved, based on the *total variation norm* which, in an N-dimensional space, is the l^∞-norm defined as $\|v\|_v^\infty = \max_{i\in\mathbb{N}_{[1,N]}} |v_i|$. As all norms in a finite-dimensional space are equivalent, the convergence result holds for any chosen norm. The error analysis, however, is vulnerable to the kind of norm used.

As it was mentioned earlier, Claim 1 does not hold for an arbitrary vector norm $\|.\|_v^\infty$ but only for such that $\|\overrightarrow{p}\|_v \leq 1$ for any distribution vector $\overrightarrow{p}$. Similarly, the norm in Claim 2 can not be chosen arbitrarily. Moreover, since the vector $\overrightarrow{p\,(i)}$ is not a distribution, i. e. we only know that for all $j \in \mathbb{N}_{[1,N]} : 0 \leq p_j\,(i) \leq 1$, the norm restrictions are even more severe. Namely, we can only use a norm such that $\|\overrightarrow{p}\|_v \leq 1$ for any vector $\overrightarrow{p}$ with $0 \leq p_j \leq 1$ for all $j \in \mathbb{N}_{[1,N]}$.

To illustrate that, let us use the Euclidean vector norm $\|.\|_v^2$. Clearly $\|\overrightarrow{p}\|_v^2 \leq 1$ for any distribution vector $\overrightarrow{p}$ and therefore for this norm the result of Claim 1 holds. On the other hand, if $|S| = N$ and $\mathcal{L}_\epsilon$ is such that $\sum_{i=0}^{\mathcal{L}_\epsilon - 1} \gamma_i(t) \leq \frac{\varepsilon}{4}$ then we have:

$$\left\| \sum_{i=0}^{\mathcal{L}_\epsilon - 1} \gamma_i(t) \cdot \overrightarrow{p\,(i)} \right\|_v^2 \leq \frac{\sqrt{N}}{4}\varepsilon, \quad \text{but } \textbf{\textit{not}} \quad \left\| \sum_{i=0}^{\mathcal{L}_\epsilon - 1} \gamma_i(t) \cdot \overrightarrow{p\,(i)} \right\|_v^2 \leq \frac{\varepsilon}{4}.$$

The latter, considering the derivations in [145], shows that the results of Claim 2 do not hold for the Euclidean norm.

The error analysis below is done for vector elements and uses the l^∞-norm $\|.\|_v^\infty$. This avoids the above mentioned problems and at the same time keeps the results practical, because the l^∞-norm is easily to compute.

3.3.1 Transient analysis

For the case $\mathcal{L}_\epsilon \leq K \leq \mathcal{R}_\epsilon$ we consider:

$$\overrightarrow{\pi^o\,(t)} = \frac{1}{W} \sum_{i=\mathcal{L}_\epsilon}^{K} w_i(t) \overrightarrow{p^o\,(i)} + \overrightarrow{p^o\,(K)} \left(1 - \frac{1}{W} \sum_{i=\mathcal{L}_\epsilon}^{K} w_i(t) \right),$$

obtained from (3.3) by replacing the lower bound of the index of the second summation by $i = \mathcal{L}_\epsilon$, as it was done in (3.5), and assuming the Fox-Glynn algorithm is used for computations. This is where $w_i(t)$ and W play a role.

Theorem 5 *Let $(S,\ \mathbf{P},\ L)$ be an aperiodic DTMC with initial distribution $\overrightarrow{p^o}$, steady-state distribution $\overrightarrow{p^{o,*}}$ and $Ind \subseteq S$. If for some K and $\delta > 0$ it holds that $\forall i \geq K$:*

$\left\|\overrightarrow{p^{o,*}} - \overrightarrow{p^o\,(i)}\right\|_v^\infty \leq \delta$ *then for*

$$\overrightarrow{\pi^{o,*}\,(t)} = \sum_{i=0}^{\infty} \gamma_i(t)\overrightarrow{p^o\,(i)}$$

and for inaccuracy $\varepsilon > 0$*:*

$$\overrightarrow{\pi^o\,(t)} = \begin{cases} \overrightarrow{p^o\,(K)} & , \text{ if } K < \mathcal{L}_\epsilon \\ \frac{1}{W}\sum_{i=\mathcal{L}_\epsilon}^{K} w_i(t)\overrightarrow{p^o\,(i)} + \overrightarrow{p^o\,(K)}\left(1 - \frac{1}{W}\sum_{i=\mathcal{L}_\epsilon}^{K} w_i(t)\right) & , \text{ if } \mathcal{L}_\epsilon \leq K \leq \mathcal{R}_\epsilon \\ \frac{1}{W}\sum_{i=\mathcal{L}_\epsilon}^{\mathcal{R}_\epsilon} w_i(t)\overrightarrow{p^o\,(i)} & , \text{ if } K > \mathcal{R}_\epsilon \end{cases}$$

the following inequality holds:

$$\boxed{\left|\sum_{j\in Ind}\left(\pi_j^{o,*}\,(t) - \pi_j^o\,(t)\right)\right| \leq 2\delta|Ind| + \frac{3}{4}\varepsilon}$$

Here W*,* $w_i(t)$*,* $\mathcal{L}_\epsilon$*, and* $\mathcal{R}_\epsilon$ *are computed using the Fox-Glynn algorithm, such that* $\sum_{i=0}^{\mathcal{L}_\epsilon - 1}\gamma_i(t) \leq \frac{\varepsilon}{4}$*, and* $\sum_{i=\mathcal{R}_\epsilon+1}^{\infty}\gamma_i(t) \leq \frac{\varepsilon}{4}$*, and* $|Ind|$ *is the cardinality of* Ind*.*

Proof See the proof of Theorem 36 from Appendix 3.3.1. □

Then the following corollary gives the error bound for the steady-state detection procedure.

Corollary 6 *Under the same conditions as Theorem 5:*

$$\boxed{\left\|\overrightarrow{p^{o,*}} - \overrightarrow{p^o\,(K)}\right\|_v^\infty \leq \frac{\varepsilon}{8|Ind|} \quad \textit{implies} \quad \left|\sum_{j\in Ind}\left(\pi_j^{o,*}\,(t) - \pi_j^o\,(t)\right)\right| \leq \varepsilon} \qquad (3.7)$$

Proof See the proof of Corollary 37 from Appendix 3.3.1. □

Let us now return to the calculation of time-bounded reachability probabilities. For computing the probability $Prob\left(s,\ \mathcal{A}\ \mathrm{U}^{[0,t]}\ \mathcal{G}\right)$ in state s, we have $Ind = \mathcal{G}$ and $\overrightarrow{p^o} = \overrightarrow{1_{\{s\}}}$. According to the above results, the safe stopping criterion to obtain an overall inaccuracy of ε equals $\left\|\overrightarrow{p^{s,*}} - \overrightarrow{p^s\,(K)}\right\|_v^\infty \leq \frac{\varepsilon}{8|\mathcal{G}|}$. Below we point out the main differences between our result and the results referred to in Section 3.1.1.

First of all, Theorem 5 takes into account the weights $w_i(t)$ (and the normalization factor W) for determining $\overrightarrow{\pi^o\,(t)}$. Hence, different summation bounds for the case $\mathcal{L}_\epsilon \leq K \leq \mathcal{R}_\epsilon$ (as $w_i(t) = 0$ for all $i < \mathcal{L}_\epsilon$) occur in the definition of $\overrightarrow{\pi^o\,(t)}$. Secondly, due to the refined bound for Fox-Glynn (cf. Proposition 4), the bounds on the left and right truncation errors on which Theorem 5 is based are two times tighter than ($\frac{1}{4}$ instead of $\frac{1}{2}$) the corresponding truncations errors that form the basis for Claim 1.

Theorem 5 refers to the l^∞-norm, whereas the norm in Claim 1 is left implicit. The resulting steady-state detection criterion:

$$\left\|\overrightarrow{p^{o,*}} - \overrightarrow{p^o\left(K\right)}\right\|_v^\infty \leq \frac{\varepsilon}{8}$$

for the case $|Ind| = 1$ (the error for a single vector element $\pi_j^o\left(t\right)$), is tighter than the bound provided in [97] (see Equation (3.4)):

$$\left\|\overrightarrow{p^{o,*}} - \overrightarrow{p^o\left(K\right)}\right\|_v \leq \frac{\varepsilon}{4}$$

The fact that the resulting bound is similar to that in Section 3.1.2 for backward computations is due to the fact that the weights introduce an additional error. In the next section, it will be shown that for backward computations—even taking into account the error introduced by weights—the steady-state detection criterion is weaker than in [143, 145].

3.3.2 Time-bounded reachability

Notice, that, unlike the case for transient probabilities, $\forall j \in \mathbb{N}_{[1,N]} : p_j^* - p_j\left(i\right) \geq 0$ for all $i \geq K$, because $\forall j \in \mathbb{N}_{[1,N]}, \forall i \geq 0 : p_j\left(i\right) \leq p_j\left(i+1\right) \leq \overrightarrow{p^*}$.

Theorem 7 *Let $(S,\ \mathbf{P},\ L)$ be an aperiodic DTMC with $Ind \subseteq S$ such that $\forall j \in Ind$: $P(j,j) = 1$, $\overrightarrow{p\left(i\right)} = \mathbf{P}^i \cdot \overrightarrow{1_{Ind}}$ and steady-state vector $\overrightarrow{p^*}$. If for some K and $\delta > 0$ it holds that $\forall i \geq K : \forall j \in \mathbb{N}_{[1,N]} : 0 \leq p_j^* - p_j\left(i\right) \leq \delta$, then for*

$$\overrightarrow{\pi^*\left(t\right)} = \sum_{i=0}^{\infty} \gamma_i(t) \overrightarrow{p\left(i\right)}$$

and for inaccuracy $\varepsilon > 0$:

$$\overrightarrow{\pi\left(t\right)} = \begin{cases} \overrightarrow{p\left(K\right)} & , \text{ if } K < \mathcal{L}_\epsilon \\ \frac{1}{W}\sum_{i=\mathcal{L}_\epsilon}^{K} w_i(t)\overrightarrow{p\left(i\right)} + \overrightarrow{p\left(K\right)}\left(1 - \frac{1}{W}\sum_{i=\mathcal{L}_\epsilon}^{K} w_i(t)\right) & , \text{ if } \mathcal{L}_\epsilon \leq K \leq \mathcal{R}_\epsilon \\ \frac{1}{W}\sum_{i=\mathcal{L}_\epsilon}^{\mathcal{R}_\epsilon} w_i(t)\overrightarrow{p\left(i\right)} & , \text{ if } K > \mathcal{R}_\epsilon \end{cases}$$

the following inequality holds:

$$\boxed{\left\|\overrightarrow{\pi^*\left(t\right)} - \overrightarrow{\pi\left(t\right)}\right\|_v^\infty \leq \delta + \frac{3}{4}\varepsilon}$$

Here W, $w_i(t)$, $\mathcal{L}_\epsilon$, and $\mathcal{R}_\epsilon$ are computed using the Fox-Glynn algorithm, such that $\sum_{i=0}^{\mathcal{L}_\epsilon - 1} \gamma_i(t) \leq \frac{\varepsilon}{4}$, and $\sum_{i=\mathcal{R}_\epsilon+1}^{\infty} \gamma_i(t) \leq \frac{\varepsilon}{4}$.

Proof See the proof of Theorem 38 from Appendix B.2.2. □

Corollary 8 *Under the same conditions as Theorem 7:*

$$\boxed{\left\|\overrightarrow{p^*} - \overrightarrow{p\left(K\right)}\right\|_v^\infty \leq \frac{\varepsilon}{4} \text{ implies } \left\|\overrightarrow{\pi^*\left(t\right)} - \overrightarrow{\pi\left(t\right)}\right\|_v^\infty \leq \varepsilon} \qquad (3.8)$$

Proof See the proof of Corollary 39 from Appendix B.2.2. □

When computing the probability $Prob\left(s,\ \mathcal{A}\ \mathrm{U}^{[0,t]}\ \mathcal{G}\right)$ we have $Ind = \mathcal{G}$ and $\overrightarrow{1_{Ind}} = \overrightarrow{1_{\mathcal{G}}}$. Recall that the main difference with the forward algorithm is that we now employ a global model-checking procedure, i.e., probabilities $Prob\left(s,\ \mathcal{A}\ \mathrm{U}^{[0,t]}\ \mathcal{G}\right)$ are determined for *all* states s. To guarantee an overall error bound of ε, one should use $\left\|\overrightarrow{p^*} - \overrightarrow{p\left(K\right)}\right\|_v^{\infty} \leq \frac{\varepsilon}{4}$ as a stopping criterion. Although our result at first sight looks quite similar to that in [145], there are various small, though important differences. As for the forward case, the influence of weights (that may yield an additional error) is taken into account. Secondly, the change of the summation lower bound from $i = 0$ to $i = \mathcal{L}_\epsilon$ in equation (3.5) is implicitly taken care of due to the fact that $\forall i < \mathcal{L}_\epsilon : w_i(t) = 0$. If weights are neglected, as in Claim 2, an error bound is obtained that is too liberal. Finally, we remark that the steady-state detection error is one-sided for backward computations.

3.3.3 Summary of results

To summarize, we provide Table 3.1 in which we include the claims presented in the works of Malhotra *et. al* and Younes *et. al*, and the corresponding (proven) results of this work. Please take into account that our results require the use of weights for the computations of $\overrightarrow{\pi^o\left(t\right)}$ and $\overrightarrow{\pi\left(t\right)}$ as well as different left and right truncation points for the computation of Poisson probabilities (cf. Proposition 4).

Forward computations
Malhotra et. al*'s result [97]:* $\left\|\overrightarrow{p^{o,*}} - \overrightarrow{p^o\left(K\right)}\right\|_v \leq \frac{\varepsilon}{4}$ implies $\left\|\overrightarrow{\pi^{o,*}\left(t\right)} - \overrightarrow{\pi^o\left(t\right)}\right\|_v \leq \varepsilon$ (3.4) *Our result:* $\left\|\overrightarrow{p^{o,*}} - \overrightarrow{p^o\left(K\right)}\right\|_v^{\infty} \leq \frac{\varepsilon}{8\lvert Ind\rvert}$ implies $\left\lvert\sum_{j\in Ind}\left(\pi_j^{o,*}\left(t\right) - \pi_j^o\left(t\right)\right)\right\rvert \leq \varepsilon$ (3.7)
Backward computations
Younes et. al*'s result [145]:* $\left\|\overrightarrow{p^*} - \overrightarrow{p\left(K\right)}\right\|_v \leq \frac{\varepsilon}{8}$ implies $\forall j \in S : -\frac{\varepsilon}{4} \leq \pi_j^*\left(t\right) - \pi_j\left(t\right) \leq \frac{3}{4}\varepsilon$ (3.6) *Our result:* $\left\|\overrightarrow{p^*} - \overrightarrow{p\left(K\right)}\right\|_v^{\infty} \leq \frac{\varepsilon}{4}$ implies $\left\|\overrightarrow{\pi^*\left(t\right)} - \overrightarrow{\pi\left(t\right)}\right\|_v^{\infty} \leq \varepsilon$ (3.8)

Table 3.1: The summary of the results

3.4 Safely detecting stationarity

Although the (theoretical) results obtained so far in this chapter provide safe criteria for detecting whether an equilibrium has been reached, they suffer from the problem that

the stopping criterion refers to the steady-state vector $\overrightarrow{p^*}$ that is typically unknown. A possible way to circumvent this is to use the absolute convergence test (see Section 3.1.1 and 3.1.2). This boils down to comparing probability vectors that are $M > 0$ iterations apart but, however, introduces an unknown error.

To avoid this unpredictable error, in case of model checking time-bounded reachability property, we suggest to exploit the typical structure of the CTMC. Recall that for checking the formula $\mathcal{A}\ \mathrm{U}^{[0,t]}\ \mathcal{G}$, all states in $\mathcal{G}$ and in $\mathcal{I} = S \setminus (\mathcal{A} \cup \mathcal{G})$ are made absorbing [8]. Intuitively speaking, on the long run, the probability mass will flow to the states in $\mathcal{G}$ and in $\mathcal{I}$, or to bottom strongly connected components (BSCCs)—SCCs that once entered cannot be left anymore—in the remainder of the CTMC, i.e., in the set of states $S \setminus (\mathcal{G} \cup \mathcal{I})$. It can be shown (see below), that we can safely replace each of these BSCCs by a single absorbing state without affecting the validity of the time-bounded reachability problem. Checking for an equilibrium now amounts to check whether the residual probability mass in the remaining non-absorbing states is below a certain threshold. Let $B_{\mathcal{A},\mathcal{G}} = \{s \in B \cap (\mathcal{A} \setminus \mathcal{G}) \,|\, B \text{ is a BSCC in } \mathbf{Q}\,[\mathcal{I} \cup \mathcal{G}]\}$.

Proposition 9 *For any state s in CTMC $(S,\ \mathbf{Q},\ L)$, time-bounded property $\mathcal{A}\ \mathrm{U}^{[0,t]}\ \mathcal{G}$ and $\mathbf{Q}^B = \mathbf{Q}\,[\mathcal{I} \cup \mathcal{G} \cup B_{\mathcal{A},\mathcal{G}}]$ we have:*

$$\mathit{Prob}\left(s,\ \mathcal{A}\ \mathrm{U}^{[0,t]}\ \mathcal{G}\right)\ \mathit{in}\ (S,\ \mathbf{Q},\ L)\ =\ \mathit{Prob}\left(s,\ S\ \mathrm{U}^{[t,t]}\ \mathcal{G}\right)\ \mathit{in}\ (S, \mathbf{Q}^B)$$

Proof See the proof of Proposition 40 from Appendix B.3. □

Every state $s \in \mathcal{A} \setminus (\mathcal{G} \cup B_{\mathcal{A},\mathcal{G}}) = S \setminus (\mathcal{I} \cup \mathcal{G} \cup B_{\mathcal{A},\mathcal{G}})$ is a transient state. This follows directly from the construction of the matrix $\mathbf{Q}^B$.

Forward computations. As the probability mass in transient states of $\mathbf{Q}^B$ on the long run equals 0, this can now be exploited. Due to the uniformization procedure the same is valid for the stochastic matrix $\mathbf{P}_B$ obtained after uniformizing CTMC $(S,\ \mathbf{Q}^B)$. When i increases, while computing $\overrightarrow{1_{\{s\}}} \cdot \mathbf{P}_B^i$, the probability to be in a transient state is only decreasing, and the probability to be in an absorbing state is increasing.

Theorem 10 *For the stochastic matrix $\mathbf{P}_B$ obtained after uniformizing CTMC $(S,\ \mathbf{Q}^B)$, for any K and $\delta > 0$ the following holds:*

$$\sum_{j \in \mathcal{A} \setminus \left(\mathcal{G} \cup B_{\mathcal{A},\mathcal{G}}\right)} p_j^s\,(K) \leq \delta \Rightarrow \forall i \geq K : \left\| \overrightarrow{p^{s,*}} - \overrightarrow{p^s\,(i)} \right\|_v^{\infty} \leq \delta$$

Where $p_j^s\,(i)$ is the j'th component of $\overrightarrow{p^s\,(i)} = \overrightarrow{1_{\{s\}}} \cdot (\mathbf{P}_B)^i$, and $\overrightarrow{p^{s,*}}$ is the steady-state probability for $\mathbf{P}_B$ when starting from state s.

Proof See the proof of Theorem 41 from Appendix B.3. □

Notice that this theorem gives a precise error bound for a steady-state detection. In particular, the premise does not refer to the steady-state prob. vector $\overrightarrow{p^{s,*}}$. Still the convergence rate is not known so the check for steady-state should be performed every M iterations as before.

Backward computations. The backward algorithm is based on $\overrightarrow{p\,(i)} = (\mathbf{P})^i \cdot \overrightarrow{1_{\mathcal{G}}}$, where vector $\overrightarrow{1_{\mathcal{G}}}$ is not a distribution. The idea of backward computations is to accumulate the probability to reach states in $\mathcal{G}$. Information about the probability reaching $\mathcal{A} \setminus (\mathcal{G} \cup B_{\mathcal{A},\mathcal{G}})$ or $B_{\mathcal{A},\mathcal{G}} \cup \mathcal{I}$ is, however, not available. To compute the precise equilibrium, we propose to compute, in addition to $\overrightarrow{p\,(i)}$, the probability to be in either $B_{\mathcal{A},\mathcal{G}}$ or $\mathcal{I}$ after i steps.

Theorem 11 *For the stochastic matrix $\mathbf{P}_B$ obtained after uniformizing CTMC $(S,\ \mathbf{Q}^B)$, for any K and $\delta > 0$ the following holds:*

$$\left\| \overrightarrow{1} - \left(\overrightarrow{p\,(K)} + \overrightarrow{p^B\,(K)} \right) \right\|_v^{\infty} \leq \delta \Rightarrow \forall i \geq K : \left\| \overrightarrow{p^*} - \overrightarrow{p\,(i)} \right\|_v^{\infty} \leq \delta$$

where $\overrightarrow{p\,(i)} = \mathbf{P}_B^i \cdot \overrightarrow{1_{\mathcal{G}}}$, $\overrightarrow{p^B\,(i)} = (\mathbf{P}_B)^i \cdot \overrightarrow{1_{B_{\mathcal{A},\mathcal{G}} \cup \mathcal{I}}}$, and $\overrightarrow{p^*} = \lim_{i\to\infty} (\mathbf{P}_B)^i \cdot \overrightarrow{1_{\mathcal{G}}}$.

Proof See the proof of Theorem 42 from Appendix B.3. □

A few remarks are in order. The premise in Theorem 11 does not refer to the (typically) unknown steady-state prob. vector $\overrightarrow{p^*}$. Moreover, the premise can be checked easily provided the two prob. vectors $\overrightarrow{p\,(K)}$ and $\overrightarrow{p^B\,(K)}$ are known. Note that two probability vectors are required to be able to detect steady state, which is similar as for using the absolute convergence test. However, premature stationarity does never occur.

3.5 Experimental results

This section reports on some experiments that we conducted with existing and the proposed approaches towards on-the-fly steady-state detection. The experiments concentrate on illustrating the phenomenon of premature stationarity in existing model checkers for CTMCs and showing the effect of the technique proposed in Section 3.4.

This is first done by means of a simple, though artificial example. The fact that these phenomena occur in realistic examples too is illustrated by means of the workstation cluster case study, and by the IEEE 802.11 group communication protocol, both of which are presented in Section 1.3. We finally report on the computation time needed for our proposed algorithm.

The tools that are used in the experiments are PRISM, E⊢MC^2 and our model checker called MRMC, all of them are discussed in Section 1.4. The first two support an on-the-fly steady-state detection as described in Section 3.1.2, whereas MRMC realizes (as an option) the steady-state detection criteria proposed in Sections 3.3.2 and 3.4. At the time of our experiments GreatSPN v1.0 [38] used E⊢MC^2 as a back-end and the results reported on E⊢MC^2 therefore were applicable to it. The latest versions of GreatSPN use MRMC.

All experiments consider the backward algorithm. For comparison reasons, probabilities obtained from Matlab or UltraSAN [120] are used. For all presented examples, curves obtained with Matlab (UltraSAN) and MRMC, with the steady-state detection turned on, coincide.

It should be noted that each tool uses different M for steady-state detection, when employing relative and absolute convergence tests (see Section 1.1.1 on page 6). For

example, PRISM uses $M = 1$, which allows to save on memory usage as only a single probability vector suffices, while E $\vdash$MC^2 uses $M = 10$. As a result, PRISM detects a steady-state earlier than E $\vdash$MC^2.

A slowly convergent CTMC. Consider the CTMC in Figure 3.1 and let $\mathcal{A} = \{0, 1\}$ and $\mathcal{G} = \{2\}$. The peculiarity of this CTMC is that the probability to move from the starting state 1 to the goal state in one or more steps is very low. At the same time it is easy to see that in the long run the probability to be in the $\mathcal{G}$ state, when starting in state 1 equals to one.

Figure 3.2 plots the probability $\mathit{Prob}\left(1,\ \mathcal{A}\ \mathrm{U}^{[0,t]}\ \mathcal{G}\right)$ for the considered tools for different time bound t. The experiments for PRISM are performed using either a relative (rel) or an absolute (abs) convergence test. As we want to make these two convergence tests behave similarly, the relative error is set to 10^{-1}. This approximately corresponds to an absolute error of 10^{-6}. Note that E $\vdash$MC^2 and both variants of PRISM (abs and rel) detect stationarity prematurely whereas MRMC does not. For the indicated range of t, the resulting error is within the inaccuracy $\varepsilon = 10^{-6}$; for larger values of t (up to around 16,000), the resulting probabilities for E $\vdash$MC^2 and PRISM differ more than ε (and MRMC, as it should be, does not detect the equilibrium).

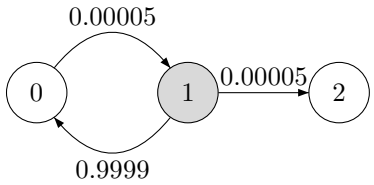


Figure 3.1: A slowly convergent CTMC

The details on the iteration index (K) at which an equilibrium is detected, and the corresponding probability are given in Table 3.5. Note that $\overrightarrow{p^*} = \lim_{K\to\infty} \mathbf{P}^K \cdot \overrightarrow{1_\Psi}$ is not a distribution but a vector of probabilities, since the backward computations are used. For this example $\overrightarrow{p^*} = (1.0, 1.0, 1.0)^T$.

To validate the tweak of the relative error bound for PRISM, it should be noted that with the original error bound 10^{-6}, the premature steady-state detection still occurs but for larger values of the time bound t, such as $t \geq 1,050,000$.

Tool	Error	K	$\mathbf{P}^K \cdot \overrightarrow{1_\Psi}$
PRISM (abs)	10^{-6}	2	$(5.00025 \cdot 10^{-5},\ 2.5 \cdot 10^{-9},\ 1.0)$
PRISM (rel)	10^{-1}	12	$(5.00275 \cdot 10^{-5},\ 2.75 \cdot 10^{-8},\ 1.0)$
E $\vdash$MC^2	10^{-6}	20	$(5.00475 \cdot 10^{-5},\ 4.75 \cdot 10^{-8},\ 1.0)$
MRMC	10^{-6}	—	—

Table 3.2: Steady-state detected on iteration K

Workstation cluster. A larger and more realistic example is the workstation cluster. The time-bounded reachability property considered is the probability to eventually reach a service level below the minimum. The investigated configuration is N=5, and the minimum QoS equals 3. The resulting CTMC has about 5,000 states. The rates of

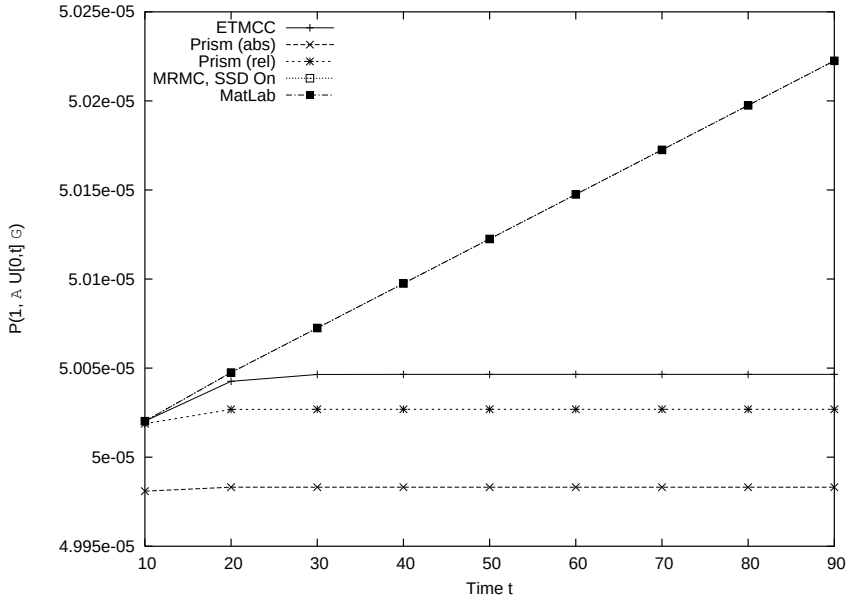


Figure 3.2: The values of $Prob\left(1,\ \mathcal{A}\ \mathrm{U}^{[0,t]}\ \mathcal{G}\right)$ for various t

the model are taken from the PRISM web page [116]. Figure 3.3 plots the computed probabilities using PRISM and E ⊢MC^2 using the absolute error 10^{-6} and relative error 10^{-3}. Here $\mathcal{A}$ contains all states whereas $\mathcal{G}$ represents the set of states for which the minimum QoS does not hold. The effect of the steady-state detection is similar as for the artificial example shown before. Note that with the default relative error 10^{-6}, PRISM prematurely detects steady state for $t \geq 28,000$.

Wireless group communication protocol. As a final example, we consider the verification of a variant of the centralized medium access protocol of the IEEE 802.11 standard for wireless local area networks. For this case study, Massink *et. al.* [100] recently reported the premature detection of steady state during probabilistic model checking. In our experiments, we confirm their results and show that our new algorithm does not suffer from these problems.

The property of interest is (as in [21, 100]) to determine whether the probability that a message originated by the AP is not received by at least one station within the duration of the time-critical phase, i. e., $t = 2.4$ seconds. Note that this time span is considered as extremely large, given that all protocol operations just last a few milliseconds on average. We consider the verification of this property for the initial state of the protocol model for different values of OD. Thus, $\mathcal{A}$ contains all states of the protocol model, whereas $\mathcal{G}$ contains all failed states, i. e., all states in which more than OD losses have taken place.

To study the influence of the steady-state detection algorithm, we use the UltraSAN model of [100] for reference purposes. We vary the omission degree OD from 0 through 8 for four number of stations in the group. The corresponding CTMC has a state space ranging from 5 to about 9,500 states. The parameters used in this case study are adopted from [100] and correspond to $PE = 0.00016$, the steady-state probability

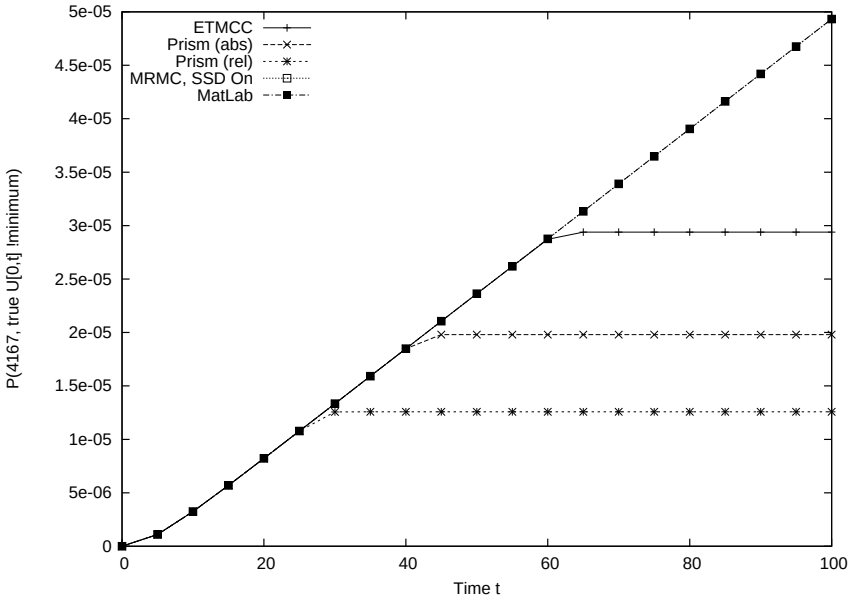


Figure 3.3: *Prob* $\left(4167,\ \mathcal{A}\ \mathrm{U}^{[0,t]}\ \mathcal{G}\right)$ for various t

to lose a message and $FDT = 0.003$, the normalized Doppler frequency caused by the relative motion of receiving and transmitting stations. Figure 3.4 plots the time-bounded probability (log-scale) versus the omission degree OD. The results of our algorithm coincide with those of UltraSAN; these results thus coincide with [100]. PRISM prematurely detects an equilibrium for all values of OD. E $\vdash$MC^2 suffers from the same phenomenon for higher omission degrees.

3.6 Time complexity and empirical evaluation

In this section we first estimate the time complexity of the forward and backward model-checking algorithms in case of the newly suggested the proposed on-the-fly steady-state detection algorithm. Then, for the case of backward computations, we report on the empirical impact of the suggested steady-state detection technique on the model-checking performance.

Time complexity. As before, we assume that $(S,\ \mathbf{Q},\ L)$ is a CTMC, with $|S| = N$ states. The number of nonzero entries in the generator matrix $\mathbf{Q}$ is D, q is the uniformization rate, and t is the time-bound of the verification property. Assume, that the sets of states $\mathcal{A}$ and $\mathcal{G}$ are given.

In case, when on-the-fly steady-state detection is not used, time complexity of computing probability *Prob* $\left(s,\ \mathcal{A}\ \mathrm{U}^{[0,t]}\ \mathcal{G}\right)$, for all initial states $s \in S$, is known to be $\mathcal{O}\left(N \cdot D \cdot q \cdot t\right)$, for *forward computations* [10], and $\mathcal{O}\left(D \cdot q \cdot t\right)$ for *backward computations* [81].

Proposed on-the-fly steady-state detection algorithms require search for BSCCs [132], which takes $\mathcal{O}\left(D\right)$ time. Choosing $\mathcal{A}$ states, belonging to BSCCs (along with

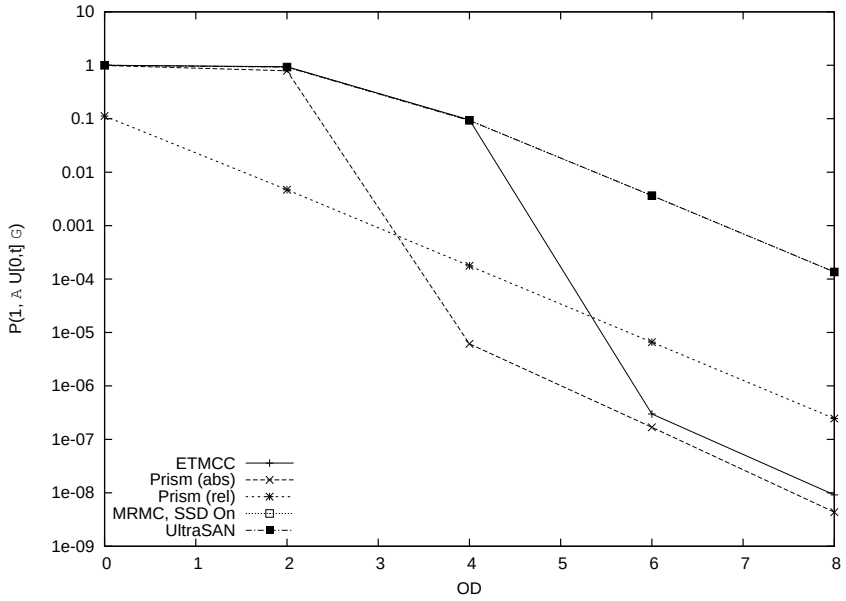


Figure 3.4: $Prob\left(1,\ \mathcal{A}\ \mathrm{U}^{[0,t]}\ \mathcal{G}\right)$ for various values of OD

making them absorbing), takes $\mathcal{O}(D)$ time. In other words, obtaining $\mathbf{P}_B$ matrix requires $\mathcal{O}(D)$ time.

For *forward algorithm* with on-the-fly steady-state detection, every M iterations the convergence criterion should be checked. It requires $\mathcal{O}(N)$ time, but the same iteration includes computing matrix vector multiplication, which overrides the influence. Thus for *forward computations*, including precise steady-state detection, the time complexity remains to be the same $\mathcal{O}(N \cdot D \cdot q \cdot t)$, if computations for all initial states s are considered.

For *backward computations* with on-the-fly steady-state detection, in addition, we have to compute $\overrightarrow{p^B(i)} = (\mathbf{P}_B)^i \cdot \overrightarrow{1_{B_{\mathcal{A},\mathcal{G}\cup\mathcal{I}}}}$, every time the $\overrightarrow{p(i)} = (\mathbf{P}_B)^i \cdot \overrightarrow{1_{\mathcal{G}}}$ is computed. This does not influence the time complexity either, thus it remains to be $\mathcal{O}(D \cdot q \cdot t)$.

As a conclusion, it is clear that for both *forward and backward algorithms*, introducing the proposed on-the-fly steady-state detection, does not change the overall time complexity of the algorithms.

Runtime. For the backward computations, the typically pattern of verification time is depicted in Figure 3.5 (These results are obtained on a *Pentium 4 3.00GHz, 2Gb RAM, Suse Linux* machine). Prior to the point at which a steady state is detected during the computation of time-bounded reachability, the run time is doubled. This is due to the fact that for the backward algorithm, $\overrightarrow{p^B(i)}$ is computed in addition to $\overrightarrow{p(i)}$. Once the equilibrium is reached (and detected), the run time for the variant with steady-state detection remains constant, whereas the run time of the algorithm without continues to grow linearly in t. Roughly speaking, if a steady-state is detected at time t', then safe on-the-fly steady-state detection is beneficial (in the sense of reducing verification time) for time spans $t \geq 2 \cdot t'$. Unfortunately, we do not know t' in advance

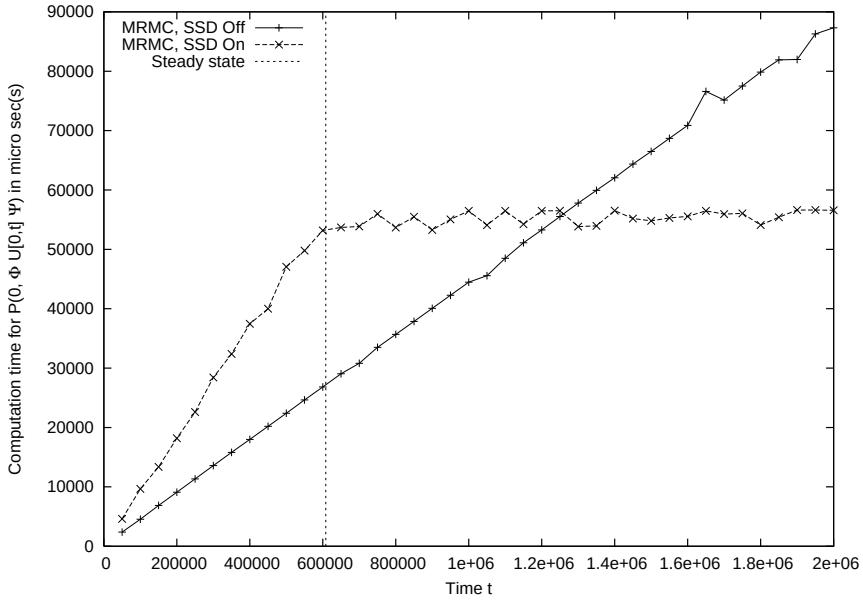


Figure 3.5: Runtime vs. time-bound t

to decide whether steady-state detection is beneficial or not.

3.7 Conclusion

In this chapter we showed that the known error bounds and convergence criteria for the standard transient analysis and time-bounded reachability algorithms, that incorporate on-the-fly steady-state detection, are seriously flawed. The latter often leads to incorrect numerical computations and thus improper model-checking results. Motivated by this fact, we first refined the error bound of the Fox-Glynn algorithm and then used it to derive the improved error bounds for the on-the-fly steady-state detection. Further, these results were complemented by a simple technique to safely detect a steady-state for the time-bounded reachability. Experiments showed that the new algorithm improves on existing techniques in probabilistic model checking. Our backward algorithm increases runtime (factor two), and requires two extra vectors. For the forward algorithm there is no increase of run time, and no additional space is required. In both cases it is guaranteed to avoid detecting equilibria prematurely.

Although for the backward algorithm the computation time is doubled (prior to reaching the steady-state, if any at time t'), and one additional probability vector of size N is needed, the computation time from approximately $2 \cdot t'$ on remains constant. For large time spans ($\geq 2 \cdot t'$), verification time is thus also reduced.

Chapter 4

Bisimulation Minimization

Like in the traditional setting, probabilistic model checking suffers from state-space explosion: the number of states grows exponentially in the number of system components and cardinality of data domains. To combat this problem, various techniques have been proposed in the literature. Variants of binary decision diagrams (multi-terminal BDDs) have been (and still are) successfully applied in PRISM to a range of probabilistic models, abstraction-refinement has been applied to reachability problems in MDPs [39], partial-order reduction techniques using Peled's ample-set method have been generalized to MDPs [53], abstract interpretation has been applied to MDPs [104], and various bisimulation equivalences and simulation pre-orders allow model aggregation prior to model checking, e. g., [15, 129]. Recently proposed techniques include abstractions of probabilities by intervals combined with three-valued logics for DTMCs [43, 71, 72] and CTMCs [85], stochastic ordering techniques for CSL model checking [98], abstraction of MDPs by two-player stochastic games [92], and symmetry reduction [90].

In this chapter we empirically investigate the effect of strong bisimulation minimization in probabilistic model checking. We hereby focus on fully probabilistic models such as DTMCs and CTMCs (cf. Section 1.1), and variants thereof with costs. The advantages of probabilistic bisimulation [93] in this setting are manifold. It preserves the validity of PCTL and CSL formulas (cf. Section 1.2). It implies ordinary lumpability of Markov chains [24], an aggregation technique for Markov chains that is applied in performance and dependability evaluation since the 1960s. Quotient Markov chains can be obtained in a fully automated way. The time complexity of quotienting is logarithmic in the number of states, and linear in the number of transitions—as for traditional bisimulation minimization—when using splay trees (a specific kind of balanced tree) for storing partitions [41]. Besides, probabilistic bisimulation can be used for obtaining (coarser) abstractions that are tailored to the properties of interest (as we will see), and enjoys the congruence property for parallel composition allowing compositional minimization. We consider explicit model checking as the non-trivial interplay between bisimulation and MTBDDs would unnecessarily complicate our study; such symbolic representations mostly grow under bisimulation minimization [65].

Thanks to extensive studies by Fisler and Vardi [45, 46, 47], it is known that bisimulation minimization for LTL model checking and invariant verification leads to drastic state space reductions (up to exponential savings) but at a time penalty: the time to

minimize and model check the resulting quotient Kripke structure significantly exceeds the time to verify the original model. In this chapter we study these issues in probabilistic (i. e., PCTL and CSL) model checking. To that end, bisimulation minimization algorithms have been realized in MRMC (cf. Chapter 2), several case studies have been considered that are widely studied in the literature, see also Chapter 1, (and can be considered as benchmark problems), and have been subjected to various experiments. Our results show that an enormous state-space reduction (up to exponential savings) may be obtained. In contrast to the results by Fisler and Vardi [45, 46, 47], the verification time of the original Markov chain mostly *exceeds* the quotienting time plus the verification time of the quotient. This effect is stronger for probabilistic bisimulation that is tailored to the property to be checked and for model checking Markov chains with costs (i. e., rewards). The latter is due to the fact that probabilistic model checking is more time-consuming than traditional model checking, while minimization w. r. t. probabilistic bisimulation is only slightly slower than for traditional bisimulation. This effect is even stronger when rewards are considered, since the verification of MRMs is rather time-consuming.

The rest of this chapter is organized as follows. Section 4.1 introduces the probabilistic bisimulation and the algorithms used. Section 4.2 presents the obtained results and their analysis. Section 4.3 concludes.

Most of the results presented below are published as [83].

4.1 Bisimulation

Let $\mathcal{D} = (S, \mathbf{P}, L)$ be a DTMC and R an equivalence relation on S. The quotient of S under R is denoted S/R. Recall that for any $C \subseteq S$ and $s \in S$:

$$\mathbf{P}(s, C) = \sum_{s' \in C} \mathbf{P}(s, s').$$

Then R is a *strong bisimulation* on $\mathcal{D}$ if for $s_1\ R\ s_2$:

$$L(s_1) = L(s_2) \quad \text{and} \quad \mathbf{P}(s_1, C) = \mathbf{P}(s_2, C) \text{ for all } C \text{ in } S/R.$$

s_1 and s_2 in $\mathcal{D}$ are strongly bisimilar, denoted $s_1 \sim_d s_2$, if there exists a strong bisimulation R on $\mathcal{D}$ with $s_1\ R\ s_2$. Strong bisimulation [24, 67] for CTMCs, that implies ordinary lumpability, is a mild variant of the notion for the discrete-time probabilistic setting: in addition to the above, it is also required that the exit rates of bisimilar states are equal: $E(s_1) = E(s_2)$.

Measure-driven bisimulation. Requiring states to be equally labeled with all atomic propositions is rather strong if one is interested in checking formulas that just refer to a (small) subset of propositions, or more generally, sub-formulas. The following notion weakens the labeling requirement in strong bisimulation by requiring equal labeling for a set of PCTL formulas F rather than for all atomic propositions. Let $\mathcal{D} = (S, \mathbf{P}, L)$ be a DTMC and R an equivalence relation on S. R is a *F-bisimulation* on $\mathcal{D}$ if for $s_1\ R\ s_2$:

$$s_1 \models \Phi \iff s_2 \models \Phi \text{ for all } \Phi \in F \quad \text{and} \quad \mathbf{P}(s_1, C) = \mathbf{P}(s_2, C) \text{ for all } C \in S/R.$$

States s_1 and s_2 are F-bisimilar, denoted $s_1 \sim_F s_2$, if there exists an F-bisimulation R on $\mathcal{D}$ with $s_1\ R\ s_2$. F-bisimulation on CTMCs (for a set of CSL formulas F) is defined analogously [11]. Note that strong bisimilarity is F-bisimilarity for $F = AP$.

Preservation results. Aziz *et al.* [6] have shown that strong bisimulation is sound and complete with respect to PCTL (and even PCTL*):

Proposition 12 *Let $\mathcal{D}$ be a DTMC, R a bisimulation and s an arbitrary state of $\mathcal{D}$. Then, for all PCTL formulas Φ, $s \models_{\mathcal{D}} \Phi \iff [s]_R \models_{\mathcal{D}/R} \Phi$.*

If for a set of PCTL formulas F, we define PCTL_F as the smallest set of formulas that contains F and is closed under all of the PCTL operators then the result above can be generalized to F-bisimulation in the following way:

Proposition 13 *Let $\mathcal{D}$ be a DTMC, R an F-bisimulation and s an arbitrary state of $\mathcal{D}$. Then, for all PCTL_F formulas Φ, $s \models_{\mathcal{D}} \Phi \iff [s]_R \models_{\mathcal{D}/R} \Phi$.*

Similar results hold for CSL and bisimulation on CTMCs [8], for PRCTL on DMRM, and for CSRL on CMRM. Note that we consider MRMs with state rewards only.

Bisimulation minimization. The preservation results suggest that one can verify properties of a Markov chain on a bisimulation quotient. The next issue to consider is how to obtain the quotient. An often used algorithm (called *partition refinement*) is based on *splitting:* Let Π be a partition of S. A splitter for some block $B \in \Pi$ is a block $Sp \in \Pi$ such that the probability to enter Sp is not the same for each state in B. In this case, the algorithm splits B into subblocks such that each subblock consists of states s with identical $\mathbf{P}(s, Sp)$. This step is repeated until a fixpoint is reached. The final partition is the coarsest bisimulation that respects the initial partition. The worst-case time complexity of this algorithm is $O(|\mathbf{P}| \log |S|)$ provided that splay trees or a data structure with a similar time-complexity of insertion and deletion (to meet this theoretical complexity bound) is used to store blocks [41]. Although they are rather memory-consuming, splay trees are adopted in our implementation.

In [40] it is reported that an implementation using red-black trees is slightly faster, although raises the time complexity to $O(|\mathbf{P}| \log^2 |S|)$. Our experiments with red-black trees did not confirm this result, but we anticipate that the reported improvement might be possible given a very efficient implementation of red-black trees. Unfortunately the latter is hard to realize due to the complexity of the data structure and operations on it.

Initial partition. The choice of initial partition in the partition refinement algorithm determines what kind of bisimulation the result is. If we group states labeled with the same atomic propositions together, the result is the strong bisimulation quotient $S/\!\sim_d$. If we choose the initial partition according to the satisfaction of formulas in F, the resulting partition is the F-bisimulation quotient $S/\!\sim_F$. To get the smallest bisimulation quotient, it is important to start with a coarse initial partition. Instead of only calculating the strong bisimulation quotient, we will also use measure-driven bisimulation for a suitable set F. In the following, we define initial partitions using sets S_0, S_1, $S_?$, U_0, and U_1 that are originally introduced in Section 1.2.1 (cf. page 12).

A naive approach for unbounded until $\mathrm{P}_{\bowtie b}(\Phi\ \mathrm{U}\ \Psi)$ is to choose $F = \{\Psi, \Phi \wedge \neg\Psi\}$. In fact, $\mathrm{P}_{\bowtie b}(\Phi\ \mathrm{U}\ \Psi)$ is not in PCTL_F, but the equivalent formula $\mathrm{P}_{\bowtie b}((\Phi \wedge \neg\Psi)\ \mathrm{U}\ \Psi)$ is. This yields an initial partition consisting of the sets $S_1 = \mathit{Sat}(\Psi)$, $S_? = \mathit{Sat}(\Phi \wedge \neg\Psi)$ and $S_0 = S \setminus (S_1 \cup S_?)$. Note that selecting $F = \{\Psi, \Phi\}$ would lead to a less efficient initial partition with four blocks instead of three. We improve this initial partition by replacing S_0 by $U_0 = \mathit{Sat}(\mathrm{P}_{\leq 0}(\Phi\ \mathrm{U}\ \Psi))$ and S_1 by U_1, which is essentially[1] $\mathit{Sat}(\mathrm{P}_{\geq 1}(\Phi\ \mathrm{U}\ \Psi))$. (Defining U_0 and U_1 as satisfaction sets of some formula has the advantage that we can still use Proposition 13.) The sets of states U_0 and U_1 can be collapsed into single states u_0 and u_1, respectively. This results in the initial partition $\{\{u_0\}, \{u_1\}, S \setminus (U_0 \cup U_1)\}$.

For time-bounded until $\mathrm{P}_{\bowtie b}\left(\Phi\ \mathrm{U}^{[0,t]}\ \Psi\right)$, one can still use U_0, but not U_1, since the fact that (almost) all paths satisfy $\Phi\ \mathrm{U}\ \Psi$ does not imply that these paths reach a Ψ-state within the time bound. Therefore the initial partition for this case is $\{\{u_0\}, \{s_1\}, S \setminus (U_0 \cup S_1)\}$ with u_0 as before and s_1 the collapsed state for S_1.

For time-interval until $\mathrm{P}_{\bowtie b}\left(\Phi\ \mathrm{U}^{[t_1,t_2]}\ \Psi\right)$ with $t_1 > 0$ we cannot use the same initial partition as for the time-bounded until. In particular the set S_1 has to be split into two parts: $\mathit{Sat}(\Phi \wedge \Psi)$ and $\mathit{Sat}(\neg\Phi \wedge \Psi)$. Remember (cf. Section 1.2.2) that paths satisfying the formula $\Phi\ \mathrm{U}^{[t_1,t_2]}\ \Psi$ should reach a state satisfying Ψ within the time span $[t_1, t_2]$ with all preceding states satisfying Φ. This means that prior to time t_1 states satisfying $\Phi \wedge \Psi$ and $\neg\Phi \wedge \Psi$ need to be distinguishable, because having a state satisfying $\neg\Phi \wedge \Psi$ before time epoch t_1 violates the formula. Thus, the initial partitioning for time-interval until is $\{\{u_0\}, \mathit{Sat}(\Phi \wedge \Psi), \mathit{Sat}(\neg\Phi \wedge \Psi), S \setminus (U_0 \cup S_1)\}$.

It is clear now that for bounded and interval until the measure-driven initial partitions are finer than for unbounded until. In the experiments reported in the next section, the effect of the granularity of the initial partition will become clear.

4.2 Experiments

To study the effect of bisimulation in model checking, we realized the minimization algorithms in MRMC and applied them to the following case studies: CP, SLE, CPS, RME, WC, WGC, P2P. The considered case studies are described in Section 1.3 and most of them can be obtained from the PRISM webpage [115].

We used PRISM to specify the models and generate the Markov chains. Subsequently, the time and memory requirements have been considered for verifying the chains (by MRMC), and for minimizing plus verifying the lumped chain (both by MRMC). All reported times are in milliseconds and are obtained by taking the average of running the experiment 10 times. The memory-usage statistics was collected the same way as it is described in Section 2.4 and again we report the peak virtual-memory usage (VSZ) in megabytes.

The time measurements were obtained on a 2.66 GHz Pentium 4 processor (32-bit) with 1 GB RAM, whereas memory usage was measured on a 2 GHz AMD dual-core processor (64-bit) with 2 GB RAM. Both machines were running Linux.

[1] Up to states s where the set $\{\sigma \in \mathit{Path}^{\mathcal{D}}(s) \mid \sigma \not\models \Phi\ \mathrm{U}\ \Psi\}$ is only almost empty.

4.2.1 Discrete time

Crowds protocol (CP) Table 4.1 summarizes the results for $\mathrm{P}_{\leq b}\,(\Diamond\, \mathit{observe})$ where *observe* characterizes a situation in which the sender's id is detected. The parameter N in the first column is the number of honest crowd members, the second column shows parameter R. The next four columns indicate the size of the state space of the DTMC (i. e., $|S|$), the number of transitions (i. e., the number of non-zero entries in $\mathbf{P}$), the verification time and the peak virtual-memory usage (VSZ) while model checking the above-mentioned property. The next three columns indicate the number of states in the quotient DTMC, the time needed for obtaining this quotient, the time to check the validity of the same formula on the quotient and the peak memory consumption while lumping and model checking. The last three columns indicate the reduction factor for the number of states, total time and VSZ.

Note that we obtain large state space reductions. Interestingly, in terms of time consumption, quotienting obtains a reduction in time of about a factor 4 to 7. The memory usage for small models was not measurable due to the insignificant execution time. For larger models the use of memory is reduced by about 30%. It is important to note that in case of F-bisimulation MRMC keeps the original probability matrix allocated and creates a new matrix for the lumped model. Therefore the reduction of used memory indicates that a significant space is needed for storing results and temporary data when model checking the unbounded-until property. Note that with the given frequency of memory sampling the model-checking phase after lumping is likely to be not covered as it takes only up to two milliseconds.

original DTMC						lumped DTMC				red. factor		
N	R	states	transitions	ver. time	VSZ	blocks	lump. time	ver. time	VSZ	states	time	VSZ
5	3	1198	2038	3.2	–	53	0.6	0.3	–	22.6	3.7	–
5	4	3515	6035	11	–	97	2.0	0.5	–	36.2	4.4	–
5	5	8653	14953	48	–	153	6.0	0.9	–	56.6	6.9	–
5	6	18817	32677	139	–	209	14	1.4	–	90	9	–
10	3	6563	15143	24	–	53	4.6	0.2	–	124	4.9	–
10	4	30070	70110	190	–	97	29	0.5	–	310	6.4	–
10	5	111294	261444	780	29.7	153	127	0.9	–	727	6.1	–
10	6	352535	833015	2640	92.1	221	400	1.4	63.4	1595	6.6	1.45
15	3	19228	55948	102	–	53	23	0.2	–	363	4.4	–
15	4	119800	352260	790	33.1	97	190	0.5	–	1235	4.1	–
15	5	592060	1754860	4670	160.6	153	1020	0.9	112.2	3870	4.6	1.43
15	6	2464168	7347928	20600	665.1	221	4180	1.5	465.1	11150	4.9	1.43

Table 4.1: CP, bisimulation: $\mathrm{P}_{\leq b}\,(\Diamond\, \mathit{observe})$

Synchronous Leader Election Protocol (SLE) The property of interest is the probability to elect a leader within N rounds: $\mathrm{P}_{\leq q}\left(\Diamond^{[0,(N+1)\cdot 3]}\, \mathit{elected}\right)$. The obtained results are summarized in Table 4.2.

For a fixed N, the number of blocks is constant. This is due to the fact that the initial state is the only probabilistic state and that almost all states that are equidistant w. r. t. this initial state are bisimilar. For $N = 4$, no gain in computation time is obtained due to the relatively low number of iterations needed in the original DTMC.

When N increases, bisimulation minimization also pays off time wise. In this case a small reduction is obtained because the time bound of the until-formula amplifies with N. The latter raises the number of matrix-vector multiplications in the model-checking procedure (cf. Section 1.2.1).

In this case the results for memory consumption are not reliable because the run times allow for at most two memory samplings per experiment (the memory is analyzed every 100 milliseconds). The present data indicates that in case of $N = 5$ and $K = 8$ the memory use is increased by about 30%, but for the larger model ($N = 4$ and $K = 16$) the increase is only about 5%.

original DTMC						lumped DTMC				red. factor		
N	K	states	transitions	ver. time	VSZ	blocks	lump. time	ver. time	VSZ	states	time	VSZ
4	2	55	70	0.02	–	10	0.05	0.01	–	5.5	0.4	–
4	4	782	1037	0.4	–	10	0.5	0.01	–	78.2	0.8	–
4	8	12302	16397	7	–	10	9.0	0.01	–	1230	0.8	–
4	16	196622	262157	165	30.9	10	175	0.01	32.3	19662	0.9	0.96
5	2	162	193	0.1	–	12	0.1	0.02	–	13.5	0.9	–
5	4	5122	6145	2.8	–	12	2.9	0.02	–	427	0.9	–
5	6	38882	46657	28	–	12	26	0.02	–	3240	1.1	–
5	8	163842	196609	140	17.7	12	115	0.02	24.8	13653	1.2	0.71

Table 4.2: SLE, bisimulation: $\mathrm{P}_{\leq q}\left(\Diamond^{[0,(N+1)\cdot 3]}\, \mathit{elected}\right)$

Cyclic Server Polling System (CPS) For this case study we model-check the formula: $\mathrm{P}_{\bowtie b}\left(\bigwedge_{j\neq 1}^{N} \neg \mathit{serve}_j \;\mathrm{U}\; \mathit{serve}_1\right)$, i. e. with probability $\bowtie\ b$ station 1 will be served before any other station. We also consider a time-bounded version thereof[2]. Note that the original model is a CTMC and thus, in order to switch to the discrete-time domain, we verify the properties on the DTMC that is obtained after uniformization.

Ordinary (strong) bisimulation yields no state-space reduction. The results for measure-driven bisimulation minimization are summarized in Table 4.3. In checking the bounded-until formula, we used the naive initial partition $\{\{s_0\}, \{s_1\}, S_?\}$. The improved initial partition with $\{u_0\}$ would have led to almost the same number of blocks as the unbounded until, e. g., 46 instead of 151 blocks for $N = 15$. For both formulas, large reductions in state space size as well as computation time are obtained; the effect of $\{u_0\}$ on the number of blocks is also considerable.

The only reliable results for the peak-memory consumption are available in case of $N = 15$. The memory usage, when model checking the unbounded-until formula with lumping, is reduced by 22%. We do not provide results for the time-bounded until due to the use of a specific initial partitioning.

Randomized Mutual exclusion (RME) Table 4.4 summarizes our results for verifying the property that process 1 is the first to enter the critical section, i. e., the PCTL formula $\mathrm{P}_{\leq q}\left(\bigwedge_{j\neq 1}^{N} \neg \mathit{enter}_j \;\mathrm{U}\; \mathit{enter}_1\right)$. Due to the relatively high number of transitions, quotienting the DTMC according to AP-bisimilarity is computationally expen-

[2] An arbitrarily chosen upper time bound was set to 1010.

original DTMC						time-bounded until				unbounded until					
						lumped DTMC		red. factor		lumped DTMC			red. factor		
N	states	transitions	$U^{[0,t]}$ time	U time	U VSZ	blocks	lump. + ver. time	states	time	blocks	lump. + ver. time	VSZ	states	time	VSZ
4	96	368	1.4	2.1	–	19	0.4	5.1	3.5	12	0.9	–	8	2.3	–
6	576	2784	10	11	–	34	1.2	16.9	8.3	18	1.4	–	32	7.9	–
8	3072	17920	62	52	–	53	4.0	58	15.5	24	2.9	–	128	17.9	–
12	73728	577536	3050	3460	25.7	103	120	716	25.4	36	55	–	2048	62.9	–
15	737280	6881280	39000	32100	269.6	151	1590	4883	24.5	45	580	210.0	16384	55.3	1.28

Table 4.3: CPS, bisimulation: The reachability properties

original DTMC					strong bisimulation						F-bisimulation					
					lumped DTMC			red. factor			lumped DTMC			red. factor		
N	states	transitions	ver. time	VSZ	blocks	time lump.	time ver.	VSZ	states	time	VSZ	blocks	lump. + ver. time	VSZ	states	time	VSZ
3	2368	8272	3	–	1123	8	1.6	–	2.1	0.3	–	233	2.9	–	10.2	1.0	–
4	27600	123883	47	–	5224	192	19	–	5.3	0.4	–	785	29	–	35.2	1.6	–
5	308800	1680086	837	91.3	18501	2880	120	69.8	16.7	0.3	1.31	2159	507	66.4	143	1.7	1.38
6	3377344	21514489	9589	1046.6	–	$> 10^7$	–	786.2	–	–	1.33	5166	7106	774.3	653	1.4	1.35

Table 4.4: RME, bisimulation: $\mathrm{P}_{\leq q}\left(\bigwedge_{j\neq 1}^{N} \neg enter_j \ \mathrm{U}\ enter_1\right)$

original CTMC							time-bounded until $[0,40]$						time-interval until $[20,40]$					
							lumped CTMC			red. factor			lumped CTMC			red. factor		
N	states	transitions	$U^{[0,40]}$ ver. time	$U^{[0,40]}$ VSZ	$U^{[20,40]}$ ver. time	$U^{[20,40]}$ VSZ	blocks	lump. + ver. time	VSZ	states	time	VSZ	blocks	lump. + ver. time	VSZ	states	time	VSZ
8	2772	12832	36	–	49	–	239	16.3	–	11.6	2.2	–	386	24	–	7.2	2.0	–
16	10132	48160	360	3.1	480	3.2	917	70	–	11.0	5.1	–	1300	96	–	7.8	5.0	–
32	38676	186400	1860	9.4	2200	9.7	3599	300	10.4	10.7	6.2	0.90	4742	430	10.3	8.2	5.1	0.94
64	151060	733216	7200	34.1	8500	35.3	14267	1810	38.4	10.6	4.0	0.89	18082	2550	38.2	8.4	3.3	1.01
128	597012	2908192	29700	132.2	33700	136.7	56819	9300	147.9	10.5	3.2	0.89	70586	12800	148.7	8.5	2.6	0.92
256	2373652	11583520	121000	523.2	143000	541.3	226787	45700	584.3	10.5	2.6	0.90	278890	60900	585.2	8.5	2.3	0.92

Table 4.5: WC, bisimulation: The reachability properties

sive, and takes significantly more time than verifying the original DTMC. However, measure-driven bisimilarity yields a quotient that is roughly an order of magnitude smaller than the quotient under *AP*-bisimilarity. Due to the coarser initial partition, this quotient is constructed rather fast. In this case, verifying the original model is more time consuming and the memory consumption is about 25% higher.

4.2.2 Continuous time

Workstation cluster (WC) In this case study the number of correctly functioning workstations determines the level of quality of service (QoS). We concentrate on model checking the following properties:

- $\mathrm{S}_{\geq 0.7}\,(\mathit{maximum})$ – In the long run, maximum QoS will be delivered in at least 70% of the cases;
- $\mathrm{P}_{\leq 0.1}\left(\Diamond^{[0,40]}\mathit{minimum}\right)$ – The probability that QoS drops below minimum, within 40 time-units, is at most 0.1;
- $\mathrm{P}_{\geq 0.9}\left(\mathit{minimum}\ \mathrm{U}^{[20,40]}\ \mathit{maximum}\right)$– The probability that QoS goes from minimum to maximum between 20 and 40 time units is at least 0.9.

The results for the steady-state property are provided in Table 4.6. The plain verifi-

original CTMC					lumped CTMC				red. factor		
N	states	transitions	ver. time	VSZ	blocks	lump. time	ver. time	VSZ	states	time	VSZ
8	2772	12832	3.6	–	1413	12	130	–	2	0.03	–
16	10132	48160	21	–	5117	64	770	2.4	2	0.03	–
32	38676	186400	114	–	19437	290	215	10.0	2	0.2	–
64	151060	733216	730	51.8	75725	1360	1670	51.8	2	0.2	1.0
128	597012	2908192	6500	202.3	298893	5900	14900	202.3	2	0.2	1.0
256	2373652	11583520	103000	801.8	1187597	25400	175000	801.8	2	0.2	1.0

Table 4.6: WC, bisimulation: $\mathrm{S}_{\geq 0.7}\,(\mathit{maximum})$

cation time of the quotient is larger than of the original CTMC, despite a state-space reduction of a factor two. This is due to the fact that model checking of the given property involves solving systems of linear equations (cf. Section 1.2.2). The convergence rate of the employed Gauss-Seidel method strongly depends on the sub-dominant eigenvalue of the iteration matrix, i. e., the closer this value is to one, the slower the convergence. In this case the sub-dominant eigenvalues of the Gauss-Seidel iteration matrix before and after lumping differ significantly. For instance for $N = 8$, the values of the original (0.156) and the quotient (0.993) are far apart and the number of iterations needed differ for about two orders of magnitude. The same applies for $N = 16$, although the differences are smaller for larger values of N. Note that the amount of memory needed for model checking with and without lumping is the same due to the low state-space reduction which is then matched with the memory consumed by storing the lumped model, the resulting partitioning and the temporary data structures needed for model checking.

The results for time-bounded and time-interval reachability are summarized in Table 4.5. These results are obtained using the measure-driven bisimulation. In contrast,

for an *AP*-bisimulation, we only obtained a 50% state-space reduction. For measure-driven bisimulation another factor 4–5 reduction is obtained. The number of blocks for time-bounded and time-interval properties differ. This is because model checking of the latter one involves a sequence of two transient analysis on different CTMCs which requires a finer initial partition (cf. Section 4.1). The reduction factors obtained for this case study are not so high, as its formal (stochastic Petri net) specification already exploits some lumping; e. g., workstations are modeled by anonymous tokens. The peak memory consumption in case of *AP*-bisimulation for the considered formulas is increased by about 8%.

Wireless Group Communication Protocol (WGC) The property of interest is, as in [100] and other studies of this protocol, the probability that a message originated by the Access Point is not received by at least one station within the duration of the time-critical phase ($t = 2.4$ milliseconds), i. e., $\mathrm{P}_{\bowtie b}\left(\Diamond^{[0,24000]} \mathit{fail}\right)$ where *fail* identifies all states in which more than *OD* losses have taken place. Table 4.7 reports the results for the verification of this property for different values of *OD* and the minimization results for a measure-driven bisimulation.

	original CTMC				lumped CTMC			red. factor		
OD	states	transitions	ver. time	VSZ	blocks	lump. + ver. time	VSZ	states	time	VSZ
4	1125	5369	121.9	–	71	13.5	–	15.9	9.00	–
12	37349	236313	7180	10.1	1821	642	10.6	20.5	11.2	0.95
20	231525	1590329	50133	61.4	10627	5431	64.9	21.8	9.2	0.95
28	804837	5750873	195086	217.3	35961	24716	231.3	22.4	7.9	0.94
36	2076773	15187833	5103900	573.1	91391	77694	611.3	22.7	6.6	0.94
40	3101445	22871849	7725041	863.5	135752	127489	922.8	22.9	6.1	0.94

Table 4.7: WGC, bisimulation: $\mathrm{P}_{\bowtie b}\left(\Diamond^{[0,24000]} \mathit{fail}\right)$

We obtain a state space reduction of about a factor 22, which results in an efficiency improvement of a factor 5 to 10. The reason that the verification times are rather excessive for this model stems from the fact that the time bound (24000) is very large, resulting in many iterations. These verification times can be improved by incorporating an on-the-fly steady-state detection, but this is not further considered here. A detailed discussion of the impact of on-the-fly steady-state detection on this case study can be found in Chapter 3. Like for the WC case study, here we have an increase of peak memory usage by about 7%.

Simple Peer-To-Peer Protocol (P2P) We verified $\mathrm{P}_{>0}\left(\Diamond^{[0,0.5]} \mathit{done}\right)$, i. e. the probability that all blocks are downloaded within 0.5 time units is greater that 0. Table 4.8 summarizes our minimization results using *AP*-bisimularity (columns 4 through 8) and the results for a recently proposed symmetry reduction technique for probabilistic systems [90] that has been realized in PRISM.
We observe that bisimulation minimization leads to a significantly stronger state-space reduction than symmetry reduction. For $N = 3$ and $N = 4$, bisimulation minimization leads to a state-space reduction of more than 23 and 66 times, respectively, the reduction of symmetry reduction. Symmetry reduction is—as expected—much faster than

original CTMC			bisimulation minimization					symmetry reduction				
			lumped CTMC			red. factor		reduced CTMC			red. factor	
N	states	ver. time	blocks	lump. time	ver. time	states	time	states	red. time	ver. time	states	time
2	1024	5.6	56	1.4	0.3	18.3	3.3	528	12	2.9	1.93	0.38
3	32768	410	252	170	1.3	130	2.4	5984	100	59	5.48	2.58
4	1048576	22000	792	10200	4.8	1324	2.2	52360	360	820	20	18.3

Table 4.8: P2P, bisimulation: $\mathrm{P}_{>0}\left(\Diamond^{[0,0.5]}\, done\right)$

bisimulation minimization, but this is a somewhat unfair comparison as the symmetries are indicated manually. These results suggest that it is affordable to first apply a (fast) symmetry reduction, followed by a bisimulation quotienting on the obtained reduced system. Unfortunately, the available tools did not allow us to test this idea, and this is left for future work.

4.2.3 Rewards

This section reports on the results for bisimulation minimization for Markov reward models. Note that the initial partitions need to be adapted such that only states with equal reward are grouped. We have equipped two DTMCs and one CTMC with a reward assignment function r:

- **CP** (DMRM): the reward indicates the number of messages sent;
- **RME** (DMRM): the reward indicates the number of attempts that have been undertaken to acquire access to the critical section;
- **WC** (CMRM): the reward is used to measure the repair time.

Recall (cf. Chapter 1) that for DMRMs, $r(s)$ indicates the reward that is earned on leaving a state, while for CMRMs, $r(s){\cdot}t$ is the earned reward when staying t time-units in s. The experiments are focused on verifying time- and reward-bounded until-formulas. For DMRMs, these formulas are checked using a path graph generation algorithm as proposed in [4] which has a time complexity in $O(k{\cdot}r{\cdot}|S|^3)$, where k and r are the time-bound and reward-bound, respectively. For CMRMs, we employed the discretization approach by Tijms and Veldman as proposed in [133] which runs in time $O(t{\cdot}r{\cdot}|S|^3{\cdot}d^{-2})$ where d is the step size of the discretization. In our experiments, the default setting is $d = \frac{1}{32}$.

For the CP case study (with $R = 3$), we checked $\mathrm{P}_{\leq 0.2}\left(\Diamond^{[0,100]}_{[0,2]}\, observe\right)$ – the probability that the sender's id is discovered, within 100 steps and maximally 2 messages sent, is at most 0.2. In case of the RME case study, we model checked the property $\mathrm{P}_{>0}\left(\bigwedge_{j\neq 1}^{N} \neg enter_j \;\mathrm{U}^{[0,50]}_{[0,10]}\; enter_1\right)$ – the probability that process one is the first to enter the critical section, withing 50 time units and 10 attempts, is greater than 0. Finally, for the WC case study, we checked $\mathrm{P}_{>0.5}\left(minimum\;\mathrm{U}^{[0,10]}_{[0,5]}\; maximum\right)$ – the probability that QoS goes from minimum to maximum, within 10 time units and

Crowds protocol with rewards										
	original DMRM				lumped DMRM			red. factor		
N	states	transitions	ver. time	VSZ	blocks	lump. + ver. time	VSZ	states	time	VSZ
5	1198	2038	2928	1.1	93	44.6	–	12.88	65.67	–
10	6563	15143	80394	2.0	103	73.5	–	63.72	1094.49	–
15	19228	55948	1004981	4.4	103	98.7	–	186.68	10182.13	–
20	42318	148578	5174951	8.9	103	161	–	410.85	32002.61	–
Randomized mutual exclusion protocol with rewards										
2	188	455	735	–	151	616	–	1.25	1.19	–
3	2368	8272	60389	1.9	1123	19010	1.5	2.11	3.18	1.27
4	27600	123883	5446685	12.7	5224	298038	6.9	5.28	18.28	1.84
5	308800	1680086	$> 10^7$	122	18501	3664530	68.6	16.69	–	1.78
Workstation cluster with rewards										
	original CMRM				lumped CMRM			red. factor		
2	276	1120	278708	1.6	147	55448	1.2	1.88	5.03	1.33
3	512	2192	849864	1.8	268	151211	1.3	1.91	5.62	1.38
4	820	3616	2110095	2.1	425	347324	1.5	1.93	6.08	1.40
5	1200	5392	$> 10^7$	3.4	618	2086575	2.1	1.94	–	1.62
6	1652	7520	$> 10^7$	4.0	847	3657682	2.5	1.95	–	1.60

Table 4.9: CP, RME and WC: Bisimulation for the reward-based properties

with at most 5 time units spent for repair, is greater than 0.5. All results are listed in Table 4.9.

Due to the prohibitive (practical) time-complexity, manageable state-space sizes are (much) smaller than for the case without rewards. Another consequence of these large verification times, bisimulation minimization is relatively cheap, and results in possibly drastic time savings, as for the Crowds protocol. The memory usage in case of lumping, as indicated in the last column of Table 4.9, is reduced. The possible memory gain is from about 21% to 46%.

4.3 Conclusion

Our experiments confirm that significant (up to exponential) state space reductions can be obtained using bisimulation minimization. The appealing feature of this abstraction technique is that it is fully automated. For several case studies, also substantial reductions in time have been obtained (up to a factor 50, cf. Table 4.3). This contrasts results for traditional model checking where bisimulation minimization typically outweighs verifying the original system. Time reduction strongly depends on the number of transitions in the Markov chain, its structure, as well as on the convergence rate of numerical computations. The P2P protocol experiment shows encouraging results compared with symmetry reduction [90] (where symmetries are detected manually). For measure-driven bisimulation for models without rewards, this speedup comes with almost no memory penalty: the peak memory use may be increased by up to 8% al-

though typically it is reduced or remains unchanged; for ordinary bisimulation some experiments showed an increase of peak memory up to 50%. In our case studies with rewards, we experienced a 21–46% reduction in peak memory use. Another important observation is that the memory needed for storing the Markov chain may not be as significant (see the CP case study) as the memory required for temporary data structures used during model checking.

Part II

Model Checking by Discrete Event Simulation

Chapter 5

Preliminaries

In the Oxford English Dictionary, simulation is described as:

> "The technique of imitating the behavior of some situation or system (economic, mechanical, etc.) by means of an analogous model, situation, or apparatus, either to gain information more conveniently or to train personnel."

In other words, simulation is the technique of building a model of a real or proposed system for studying its behavior under specific conditions. One of the key powers of simulation is the ability to model the behavior of a system as time progresses. In discrete event simulation, see the books of Mitrani [101] and Cassandras [27], the operation of a system is modeled as a chronological sequence of events. Each event occurs at an instant in time and marks a change of state in the system. By the system, we mean a collection of entities (e. g., people, machines or network services) that interact over time. The particular nature of the system and the properties we wish to understand can vary. The unknown properties of the system are estimated from the system's observed behavior. Typically, these properties represent measures of system performance that can be expressed as the mean values of some random variables.

There are numerous examples of the use of discrete event simulation in areas such as service industries, manufacturing, and office environments. An attractive feature of simulation in general is that it can be applied to both finite- and infinite-state systems. Therefore, one of the most typical application areas for discrete event simulation is analysis of queuing systems. Below, we provide a trivial example of such a system and point out the measures of interest that can be obtained using simulation. Note that this example will be used throughout the chapter for illustrative purposes.

Example 8 *Consider a bank which opens at time t_o. By this time some customers may be already waiting for the bank to open, and then during the day people come and leave once they have been served. The number of customers at the beginning of the next day depends on the number of people that were in the bank by the closing time t_c. The number of customers S_t in the bank at any time t is bounded by C_{max}.*

In this setting one may be interested in α_M^c – the probability of having C_{max} customers in the bank at time t_c. The desired value can be computed as $\alpha_M^c = E\left[I_M\left(S_{t_c}\right)\right]$,

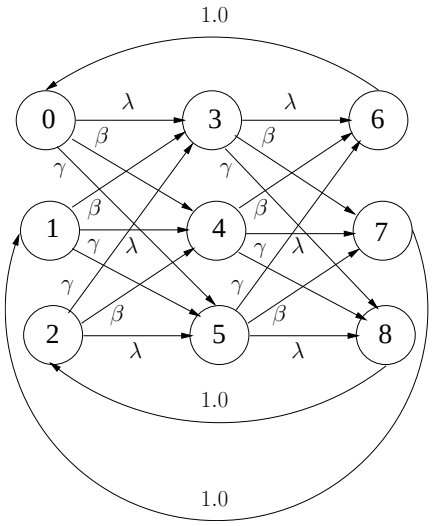


Figure 5.1: The bank system for $t_o = 8.00am$, $t_c = 10.00am$, and $C_{max} = 4$.

where the function $I_M(n)$ *is defined as follows:*

$$I_M(n) = \begin{cases} 1 & \text{if } n = C_{max} \\ 0 & \text{otherwise} \end{cases}$$

Another measure of interest is α_M *– the probability of having* C_{max} *customers in the bank at any particular work hour, i. e. in the long run. If* S_∞ *is the* r. v. *representing the number of customers in the bank in the long-run, then the value of interest can be computed as* $\alpha_M = E[I_M(S_\infty)]$.

Typically the distributions of S_{t_c} *and* S_∞ *are unknown, and thus* α_M^c *and* α_M *cannot be computed. Therefore they have to be estimated by observing the bank, e. g., we can see that on Monday* $S_{t_c} = 3$*, on Tuesday* $S_{t_c} = 1$*, on Wednesday* $S_{t_c} = 3$ *and so on.*

Figure 5.1 shows an example of such a bank modeled as a homogeneous Markov Chain. Here states $\{0, 1, 2\}$*,* $\{3, 4, 5\}$*, and* $\{6, 7, 8\}$ *correspond to customers waiting in the bank at* 8.00*,* 9.00*, and* 10.00 *o'clock respectively. States* $\{0, 3, 6\}$*,* $\{1, 4, 7\}$*, and* $\{2, 5, 8\}$ *correspond to* 0*,* 1*, and* 2 *customers being in the bank. This example we treat both as a DTMC and as a CTMC. In the former case the transition values* λ*,* β*, and* γ *are probabilities, and can be assigned any positive real values between* 0 *and* 1*, such that* $\lambda + \beta + \gamma = 1$*. In the latter case* λ*,* β*, and* γ *represent rates, and can be assigned any finite positive real value.*

Our goal is to apply discrete event simulation to model checking CSL properties on finite-state continuous-time Markov chains. In particular, we are interested in model checking probabilistic operators such as: steady-state, unbounded-until, and time-interval until. Similar to numerical model checking thereof, see Chapter 1, our approach will boil down to estimating transient and steady-state probabilities of CTMCs. In Chapter 6, among other things, we show how these probabilities can be represented as mean values of certain random variables. The latter implies that in order to apply simulation to CSL model checking we need to know how to do two things: *(i)* estimate

mean values of random variables using simulations; *(ii)* simulate random variables whose mean values represent our measures of interest. In this chapter, we are going to discuss these key points of discrete event simulation using theory (mostly) adopted from [101].

The rest of the chapter is organized as follows. We begin with Section 5.1 where we briefly explain simulation of random variables. This is done by recalling the notion of a random variable and introducing terms such as *observation* and *sample*. Further, in Section 5.2, we talk about *point estimates*. The latter are single-value estimates of a desired measure that is the parameter of a random variable. Motivated by our goal, we concentrate on the case when the measure of interest is the mean value. *Confidence intervals* are another, more precise, way to estimate parameters of random variables. Therefore, in Section 5.3 we discuss (symmetric) confidence intervals for mean values. Sections 5.4 through 5.6 are devoted to simulation methods that allow to derive confidence intervals for various measures of interest. First, we describe *terminating simulation* that is a simulation in which the desired measure of system performance is defined relative to the time interval marked by some events. Then, we discuss *steady-state simulation* where the measure of interest is defined in the limit of time going to infinity. Finally, we study a discrete-time method for simulating CTMCs that allows to estimate steady-state measures using simulation runs performed on the embedded DTMCs. The introduction to discrete event simulation ends with Section 5.7. In this section, we provide confidence intervals for the mean values of Bernoulli distributed random variables and talk about the strong law of large numbers for Bernoulli trials.

5.1 Simulating random variables

From probability theory we know that a random variable (*r. v.*) X defined on a probability space $(\Omega, \mathcal{F}, Prob)$ is a measurable function $X : \Omega \to \mathbb{R}$. Recall that $\mathcal{F}$ is a σ-algebra defined on the subsets of Ω, where every element of $\mathcal{F}$ is called an event and has a probability value associated with it via the probability measure function *Prob*. The following example shows the way of formally defining a *r. v.* and the corresponding probability space. Note that this knowledge is crucial for understanding Section 5.7 where we consider Bernoulli trials.

Example 9 *For instance consider an experiment in which we toss a fair coin. The set of possible outcomes of the experiment gives us the sample space* $\Omega = \{H, T\}$*, where* H *stands for "heads" and* T *for "tails". The* σ*-algebra is then defined as* $\mathcal{F} = \{\emptyset, \{H\}, \{T\}, \{H, T\}\}$ *and the probability measure is given by* $Prob(\{H\}) = 0.5$ *and* $Prob(\{T\}) = 0.5$*.*

For such an experiment we can determine a r. v. *as the following function of* $\omega \in \Omega$*:*

$$X^c(\omega) = \begin{cases} 1 & \text{if } \omega = H \\ 0 & \text{if } \omega = T \end{cases}$$

Note that for any $r \in \mathbb{R}$ *the set* $\{\omega \in \Omega | X^c(\omega) \leq r\}$ *is an element of the* σ*-algebra* $\mathcal{F}$*, i. e. is an event. Therefore,* X^c *is a measurable function.*

When simulating a *r. v.* X, all we normally know is the sample space of the experiment and the way the *r. v.* is defined. The latter allows us to observe the value

of X obtained from a single experiment and this value we call *an observation*. Taking M independent instances of the experiment we observe the values of M independent and identically distributed random variables (*i. i. d. r. v.*) $X_1, \ldots, X_M$, which provide us with a vector of M observations, that we call *a sample* and denote as $\overrightarrow{\mathbf{X}} = (X_1, \ldots, X_M)$.

It is important to note that further a sample is seen in two ways, first as a vector of values sampled from the *i. i. d. r. v.* $X_1, \ldots, X_M$ and second as a vector of the *i. i. d. r. v.* $\overrightarrow{\mathbf{X}}$. Normally the distinction is clear from the context or is pointed out but to avoid misinterpretation, we provide a simple rule of thumb, namely: *When it comes to examples and practical applications a sample is a vector of particular sampled values but when it is used in theoretical derivations it is typically treated as a vector of* r. v.

5.2 Point estimates

Let X be a *r. v.* and α an unknown value of the desired quantity that is a parameter of X, for instance α is the mean value of X. We would like to estimate the value of α by observing the *r. v.*, i. e. considering a sample of its observations $\overrightarrow{\mathbf{X}} = (X_1, \ldots, X_M)$. Given the sample of observations and a real-valued function $A : \mathbb{R}^M \to \mathbb{R}$, let us call $A = A(\overrightarrow{\mathbf{X}})$ a *point estimate* (*p. e.*) for the unknown value α. Generally speaking, a point estimate is a *r. v.* because it can be seen as function of the sequence of *r. v.* given by the sample $\overrightarrow{\mathbf{X}}$. So far, the *p. e.* of α was defined as an arbitrary function. The following definitions identify two important properties that a "good" *p. e.* might be expected to possess.

Definition 15 *A point estimate A for α is* unbiased *if $E[A] = \alpha$.*

Definition 16 *A point estimate A for α is* consistent *if for every $\varepsilon > 0$:*

$$Prob\left(|A - \alpha| < \varepsilon\right) \to 1 \text{ when } M \to \infty.$$

Clearly, being unbiased and consistent are two desirable properties of an estimate. The following proposition gives us a *p. e.* that has these properties for the case when the unknown parameter of X is its mean value.

Proposition 14 *If $\alpha = E[X]$, then for a sample $\overrightarrow{\mathbf{X}} = (X_1, \ldots, X_M)$ of* independent *observations,*

$$\overline{X} = \frac{1}{M} \cdot \sum_{i=1}^{M} X_i \tag{5.1}$$

is an unbiased and consistent p. e. *for α.*

It is clear that the larger the number of observations M, the better $\overline{X}$ approximates the value of α. Yet M has to stay finite and then still even an unbiased and consistent estimate can be widely off the mark on a particular sample, as is shown in the next example.

Example 10 *For the DTMC model of the bank, discussed in Example 8, let us choose* $\lambda = 0.1$, $\beta = 0.3$, *and* $\gamma = 0.6$. *Observing the* r. v. $I_M(S_{t_c})$[1], *we can have the following sample of* 10 *observations:* $(0, 1, 1, 1, 0, 1, 1, 0, 1, 0)$. *Here we assume that for every simulation run we start in the state* 0 *i. e. there are no customers waiting for the bank to open in the morning. The parameter of* $I_M(S_{t_c})$ *we want to know is its mean* α_M^c. *Then using Proposition 14, the* p. e. *of* α_M^c *is:*

$$\overline{X} = \frac{1}{10}(0 + 1 + 1 + 1 + 0 + 1 + 1 + 0 + 1 + 0) = 0.6.$$

The exact value of $\alpha_M^c = 0.3$, *which is rather different from the estimated value.*

As we see in Example 10, sometimes having a *p. e.* of an unknown parameter α of X is not enough, simply because it is not clear how close the estimate is to the real value. In this case the confidence interval approach is used, that gives an estimated range of values which is likely to include α. The estimated range is again calculated from a given sample data. This technique is discussed in the next subsection.

5.3 Confidence intervals

Generally speaking, a confidence interval is an interval in which a measurement or trial falls corresponding to a given probability. For a *r. v.* X and its unknown parameter α, the confidence interval is defined as follows:

Definition 17 *The* confidence interval (*c. i.*) *is defined by two real-valued functions of the sample* $\overrightarrow{\mathbf{X}} = (X_1, \ldots, X_M)$, *denoted* $A_l\left(\overrightarrow{\mathbf{X}}\right)$ *and* $A_r\left(\overrightarrow{\mathbf{X}}\right)$, *such that regardless of the value of* α:

$$Prob\left(A_l\left(\overrightarrow{\mathbf{X}}\right) \leq \alpha \leq A_r\left(\overrightarrow{\mathbf{X}}\right)\right) = 1 - \beta, \qquad for\ some \qquad 0 < \beta < 1.$$

The probability $1 - \beta$ *is called the* confidence *of the confidence interval.*

Confidence intervals are frequently used. In the literature one can often find 95% *c. i.*, i. e., derived with 0.95 confidence. It implies that, under repeated sampling, the obtained *c. i.* contain the mean value in 95% of cases. In other words, the confidence is a measure of the assurance you have that the derived interval contains the actual mean value.

In what follows, we shall concentrate on deriving the *c. i.* of α that is the mean of *r. v.* X ($\alpha = E[X]$). In this work we only consider *symmetric* confidence intervals, i. e. the ones that are symmetrically placed around the mean.

The rest of this section is organized as follows. First, in Section 5.3.1, we discuss the way of deriving the most general *c. i.* for α, with no assumptions on X. Further, in Section 5.3.2, we consider a special case when X is a normally-distributed *r. v.*. Under this assumption the standard confidence interval for α will be proven to have a better accuracy. Later, in Section 5.3.3, we discuss the influence of the sample size and the confidence on the width of the *c. i.*. The discussion is concluded by Section 5.3.4 where we provide an example that shows how the *c. i.* can be applied to improve the estimate of the measure given in Example 10.

[1] In this case it takes value 1 if there are two customers in the bank at 10.00 o'clock and 0 otherwise.

5.3.1 The standard confidence interval

The general approach to finding the *c. i.* is based on the existence of a *r. v.*, $Z\left(\overrightarrow{\mathbf{X}}, \alpha\right)$, a function of the sample and the unknown parameter α, whose distribution is fixed and known. This function can be derived with the help of the Central Limit Theorem.

Theorem 15 (Central Limit Theorem) *Let $X_1, X_2, \ldots$ be a sequence of* i. i. d. r. v. *with finite mean α and variance $\sigma^2 \neq 0$. Then the distribution of the* r. v.

$$\frac{\sum_{i=1}^{M} X_i - M \cdot \alpha}{\sqrt{M} \cdot \sigma} \tag{5.2}$$

approaches the standard normal distribution, $N(0,1)$, as $M \to \infty$.

The function (5.2) is a *r. v.* for which we know the distribution[2] when M goes to infinity. We also hope that for a sufficiently large value of M, we are close to this distribution. The latter assumption allows us to reason about the value of α by deriving a *c. i.* in the form of Definition 17 (see above). A similar, but sometimes more convenient form of Theorem 15, is presented below.

Theorem 16 (Central Limit Theorem, Rephrased) *Let $X_1, X_2, \ldots$ be a sequence of* i. i. d. r. v. *with finite mean α and variance $\sigma^2 \neq 0$. Then the distribution of the* r. v.

$$\frac{\sum_{i=1}^{M} X_i - M \cdot \alpha}{\sqrt{M}}$$

approaches the normal distribution, $N(0, \sigma^2)$, as $M \to \infty$.

It is important to note that Theorems 15 and 16 apply to discrete as well as continuous *r. v.* In the discrete case, $\overline{X}$ is a discrete *r. v.* and we are confronted with a series of discrete *r. v.* corresponding to different values of M. As M grows, the density function of the elements of this series converges towards a density function of a continuous variable (namely the normal distribution). This means that if we build a density function histogram for $\overline{X}$, the curve that joins the centers of the upper faces of the rectangles forming the histogram converges towards a Gaussian curve as M approaches infinity.

Now, let us derive a *c. i.* for α. By Theorem 15, it follows that the *r. v.*

$$\widetilde{Z}\left(\overrightarrow{\mathbf{X}}, \alpha\right) = \frac{\overline{X} - \alpha}{\sigma/\sqrt{M}} \tag{5.3}$$

approaches the standard normal distribution, $N(0,1)$, as $M \to \infty$. Then, for a given $\beta \in \mathbb{R}_{(0,1)}$ we can choose a non negative value $\widetilde{z_n}(\beta)$ (index n stands for *normal*) based on the limiting distribution $N(0,1)$, such that:

$$Prob\left(N(0,1) \leq \widetilde{z_n}(\beta)\right) = 1 - \frac{\beta}{2}. \tag{5.4}$$

Then, for sufficiently large M, the following holds:

$$Prob\left(\widetilde{Z}\left(\overrightarrow{\mathbf{X}}, \alpha\right) \leq \widetilde{z_n}(\beta)\right) \approx 1 - \frac{\beta}{2}. \tag{5.5}$$

[2] Even though the distribution of $X_1, X_2, \ldots$ is not known.

Let us note that, in this work, we do not dwell on the dependency between the quality of the provided approximation and sample size M. This topic is thoroughly discussed in the literature, and more information on it can be found in [44, 77].

At this point, Equation (5.5) provides us with the formula that resembles a half of the *c. i.* However, using the fact that $N(0,1)$ is a symmetric distribution, and substituting $\widetilde{Z}\left(\overrightarrow{\mathbf{X}}, \alpha\right)$ by Equation (5.3), it can be transformed into:

$$Prob\left(\overline{X} - \frac{\widetilde{z_n}(\beta)\cdot\sigma}{\sqrt{M}} \leq \alpha \leq \overline{X} + \frac{\widetilde{z_n}(\beta)\cdot\sigma}{\sqrt{M}}\right) \approx 1-\beta. \tag{5.6}$$

Equation (5.6) gives an approximation of the *c. i.* where equality is reached only when M goes to infinity. The only problem now is the parameter σ: it is generally unknown and is usually estimated using an *unbiased estimate*, given by:

$$\overline{V} = \sqrt{\frac{1}{M-1}\sum_{i=1}^{M}(X_i - \overline{X})^2}, \tag{5.7}$$

where $\overline{V}^2$ is called the *sample variance*. Using this estimate, the next approximation of the *c. i.* for α can be given as:

$$Prob\left(\overline{X} - \frac{\widetilde{z_n}(\beta)\cdot\overline{V}}{\sqrt{M}} \leq \alpha \leq \overline{X} + \frac{\widetilde{z_n}(\beta)\cdot\overline{V}}{\sqrt{M}}\right) \approx 1-\beta. \tag{5.8}$$

In general, equality in Equation (5.8) is not reached even when $M \to \infty$. The latter is because the value of σ is substituted with a non-consistent estimate $\overline{V}$. However, the provided approximation of the *c. i.* is known to be sufficiently accurate for large values of M. We should stress that in this dissertation we do not discuss the quality of approximations like the one given by Equation (5.8). The required information about this matter can be obtained from the text books which were referenced earlier.

In the next part of this section, we consider a special case when we can obtain an exact *c. i.* that has a form of Equation (5.8). The sufficient condition for that is to have *i. i. d. r. v.* $X_1, \ldots, X_M$ which are normally distributed.

5.3.2 Normally-distributed random variables

Let X be a normally distributed *r. v.* and $\overrightarrow{\mathbf{X}} = (X_1, \ldots, X_M)$ be a sample of its observations. Then we can use the following lemma to turn Equation (5.8) into equality.

Lemma 17 *If* i. i. d. r. v. $X_1, \ldots, X_M$ *are normally distributed, the* r. v.

$$\frac{\overline{X} - \alpha}{\overline{V}/\sqrt{M}}$$

has the Student's distribution with $M-1$ *degrees of freedom, denoted* t_{M-1}.

Lemma 17 states that, in case of normally distributed *r. v.* and $\overline{V}$ used instead of σ, *r. v.* $\widetilde{Z}\left(\overrightarrow{\mathbf{X}}, \alpha\right)$ has the Student's distribution. Thus, by taking $\widetilde{z_s}(\beta)$, such that:

$$Prob\left(t_{M-1} \leq \widetilde{z_s}(\beta)\right) = 1 - \frac{\beta}{2}, \tag{5.9}$$

and using it instead of $\widetilde{z_n}(\beta)$ we avoid approximation (even for finite M) and turn Equation (5.8) into equality.

Note that these results are important because of two reasons: *(i)* they are used for deriving a *c. i.* in steady-state simulation, see Section 5.5; *(ii)* for practical applications, the sample size M is always taken to be finite.

5.3.3 The width of the confidence interval

The width of a *c. i.* is something we would like to keep as small as possible in order to reduce the range of possible values for α. Equation (5.6) indicates that for a given confidence, the larger the sample size M, the smaller is the *c. i.*. Another way to shrink the *c. i.* is reducing the variance σ, for instance by choosing a better sample variance than the one given by Equation (5.7).

It is also important to note the dependency of the *c. i.* on $\widetilde{z_n}(\beta)$ (and $\widetilde{z_s}(\beta)$). It is clear from Equation (5.4) that $\widetilde{z_n}(\beta) = \Phi^{-1}\left(1 - \frac{\beta}{2}\right)$, where Φ is the cumulative density function of the $N(0,1)$ distribution, also known as the *probit* function. Figure 5.2 shows the behavior of $\widetilde{z_n}(\beta)$ versus $1 - \frac{\beta}{2}$. Notice that for $\widetilde{z_n}(\beta) \geq 0$ we should have $1 - \beta \geq 0$ and $\widetilde{z_n}(\beta)$ approaches $+\infty$ as β goes to zero. The latter means that the closer $1 - \beta$ is to one, the faster the value of $\widetilde{z_n}(\beta)$ is increasing. This implies that even a small increase of a sufficiently large confidence can significantly widen the *c. i.* The Student's t_{M-1} distribution is symmetric like the normal distribution and approaches it when $M \to \infty$. Therefore the behavior of $\widetilde{z_s}(\beta)$ is similar to $\widetilde{z_n}(\beta)$.

Now, when the main concepts of the *c. i.* have been discussed, let us move to the example.

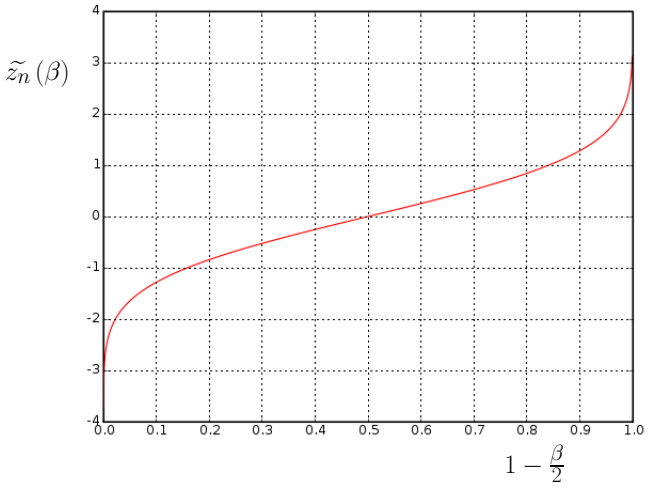


Figure 5.2: The dependency of $\widetilde{z_n}(\beta)$ from $1 - \frac{\beta}{2}$

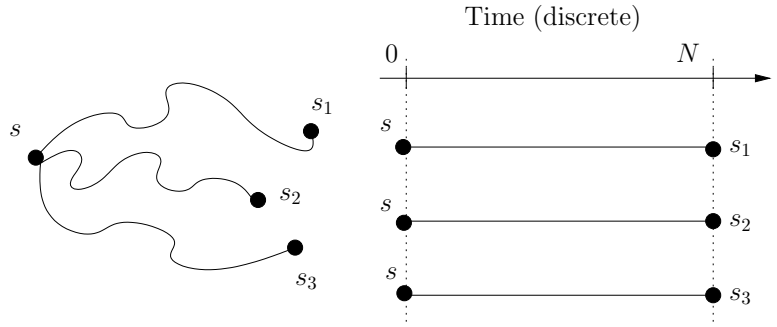


Figure 5.3: Terminating simulation until N steps

5.3.4 An example

In this section, we conclude the discussion about the *c. i.* by illustrating the use thereof with the help of the next example.

Example 11 *Let us derive a* 95% c. i. *for the mean of the* r. v. $I_M(S_{t_c})$ *from Example 10. We shall take the same sample as before, for which the* p. e. *of* α_M^c *is known to be* 0.6. *The true variance value of* $I_M(S_{t_c})$ *is unknown, the sample variance* $\overline{V}$ *computed from the sample is:*

$$\overline{V} = \sqrt{\frac{1}{10-1} \cdot (6 \cdot (1-0.6)^2 + 4 \cdot 0.6^2)} = \frac{2}{\sqrt{15}}.$$

The r. v. $I_M(S_{t_c})$ *is not known to be normally distributed, and thus we choose* $\widetilde{z_n}(\beta)$ *as for Equation (5.5). Our confidence equals* 0.95, *thus* $\beta = 0.05$ *and, like for Equation (5.4), we get:*

$$Prob\,(N(0,1) \leq \widetilde{z_n}(0.05)) = 0.975,$$

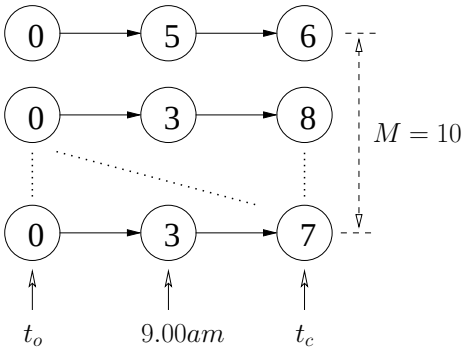
which is satisfied for $\widetilde{z_n}(0.05) = \frac{49}{25}$. *Using Equation (5.8) we obtain that with approximately* 95% *confidence,* α_M^c *belongs to the interval:*

$$\left[0.6 - \frac{\frac{49}{25} \cdot \frac{2}{\sqrt{15}}}{\sqrt{10}},\ 0.6 + \frac{\frac{49}{25} \cdot \frac{2}{\sqrt{15}}}{\sqrt{10}}\right],$$

that results in $[0.27993334, 0.92006666]$. *We know that* $\alpha_M^c = 0.3$ *and thus the resulting interval is correct. Note that in order to reduce the width of the* c. i. *we can increase the number of samples* M. *This will likely improve the value of the* p. e., *since it is consistent.*

5.4 Terminating simulation

Terminating simulation is a simulation in which the desired measures of system performance are defined relative to the interval of simulated time $[0, t_e]$ where t_e is the instant in the simulation when a specified event e occurs. The desired event e can be for example a specific time instant N, see Figure 5.3, of reaching a certain system



#run	S^{t_o}	S^{t_m} [a]	S_{t_c}	$I_M(S_{t_c})$
1	0	2	*0*	**0**
2	0	0	*2*	**1**
⋮	⋱	⋱	⋱	⋮
10	0	0	*1*	**0**

[a] Here we have $t_m = 9.00am$.

Figure 5.4: Customer observations obtained by simulating the bank model

state. The *c. i.* approach can be applied to terminating simulations in a straightforward manner. Consider the following example.

Example 12 *In Example 11 the confidence interval for* α_M^c[3] *is derived. The analysis is done based on the sample of observations of the random variable* $I_M(S_{t_c})$ *given in Example 10. So far we did not explain how this sample is obtained. Now it is time to reveal that this can be done by applying terminating simulations.*

Imagine that every morning at t_o *there are no customers waiting for the bank to open*[4]*. We observe how customers come and leave the bank until it closes. The latter can be done by simulating our DTMC model, starting in state* 0*. The bank closing indicates the end of a simulation run, i.e. it is the event that defines the interval of simulated time* $[t_o, t_c]$ *for terminating simulation. Figure 5.4 shows the simulation runs on the bank DTMC (the figure on the left), and the obtained observations (the table on the right). Recall that* $I_M(S_{t_c})$ *is a function of the random variable* S_{t_c}*, so by observing the latter one we easily get the observations for the former one.*

5.5 Steady-state simulation

In steady-state simulation, the measures of interest are defined as limits, as the length of the simulation goes to infinity. There is no natural event to terminate the simulation, so the length of the simulation is made large enough to get "good" estimates of the quantities of interest. Steady-state simulation generally poses two problems:

1. The existence of a transient phase may cause the estimate to be biased.
2. The simulation runs are long, and normally one cannot afford to carry out many independent simulations.

These are several methods that allow to cope with these problems to some extent. Among them are: the method of *independent replicas*, the *batch means method*, and the *regeneration method*, see [59, 136]. Each of these methods has its advantages and disadvantages. We have chosen to discuss, and later use, the *regeneration method*

[3] The probability of having $C_{max} = 2$ customers in the bank by its closing time at $t_c = 10.00am$.
[4] As it is assumed in Example 10.

because it allows to obtain purely independent simulation runs with omitted transient phase. Although regenerative simulations, when implemented, may not be as efficient as, say, the *batch-means method*.

Suppose that in the course of a simulation run one can identify a system state s_0 and moments of time $t_0, t_1, \ldots$, which have the following properties, for all $i \in \mathbb{N}$:

I. The distance between the consecutive time moments, $d_i = t_i - t_{(i-1)}$ is an *i. i. d. r. v.*;

II. The system state at time t_i, i. e. X_{t_i}, equals s_0;

III. The behavior of the system after time t_i depends only on the state X_{t_i}.

Then we can define the, so called, regeneration points as follows:

Definition 18 *The moments of time t_i satisfying the conditions* **I.**, **II.**, **III.**, *with $t_0 = 0$ and $t_i = min\{t > t_{(i-1)} \mid t \in \mathbb{R} \wedge X_t = s_0\}$ for all $i \in \mathbb{N}_{\geq 1}$, are called the* regeneration points*; the intervals between them are referred to as* regeneration cycles*.*

Let us illustrate the notion of the regeneration points by the following example.

Example 13 *In Example 8, we consider the bank modeled as a DTMC. The system state thus corresponds to the state of the Markov chain. We can identify the moments of time $t_0, t_2, \ldots$ as the moments when there are zero customers in the bank by the time it opens. All these time points correspond to the state* 0 *of the DTMC, and therefore the property* **II.** *is satisfied. Markov chains have a memoryless property which means that the behavior after time t_i depends only on the state occupied at this time. The latter ensures that the property* **III.** *holds as well. By our choice the state occupied at time t_i is* 0 *for any $i \in \mathbb{N}$. This makes d_i to be* i. i. d. r. v.*, hence ensuring* **I.** *It is clear now that the state* 0 *identifies the regeneration points for our simulation process.*

Even when regeneration points exist, the parameters of a system may be such that it never reaches them. It only makes sense to talk about the system being regenerative when an equilibrium exists; then the regenerative cycles are almost surely finite, meaning that d_i, for all $i \in \mathbb{N}_{\geq 1}$, is finite with probability one. The regenerative method thus can be safely applied in case of an irreducible Markov chain with finitely many states.

Example 14 *The DTMC shown in Example 8 is irreducible and has a finite number of states. This implies that a unique steady-state behavior exists. The latter implies that the regeneration points defined in Example 13 are reachable and that the regenerative cycle lengths d_i are* a.s. *finite.*

Consider an irreducible Markov chain $\{X_t \mid t \in \mathbb{R}_{\geq 0}\}$, with a finite state space S. Then for a real-valued function $f : S \to \mathbb{R}$, the value $\alpha = E[f(X)]$ has to be estimated, where $X_t \xrightarrow[t \to \infty]{a.\,s.} X$. To apply the regenerative method, the system is simulated for a number of regeneration cycles. The simulation starts and ends at a regeneration point, say state s_0 in Figure 5.5, going through a sequence of $M+1$ such points: $t_0 = 0, t_1, t_2, \ldots, t_M$. Notice that $d_i = t_i - t_{(i-1)}$ (for $i > 0$) are *i. i. d. r. v.* Let

$$Y_i = \int_{t_{(i-1)}}^{t_i} f(X_t)\, dt. \tag{5.10}$$

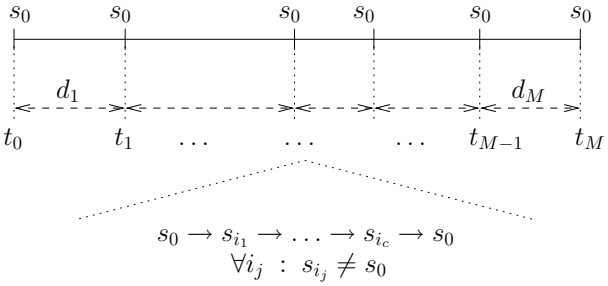


Figure 5.5: Regenerative method

(Y_1, d_1)	(Y_2, d_2)	(Y_3, d_3)	(Y_4, d_4)	(Y_5, d_5)
$(1, 3)$	$(2, 9)$	$(0, 3)$	$(3, 9)$	$(1, 6)$

Table 5.1: (Y_i, d_i) values of the bank example, DTMC

Clearly, Y_i is the accumulated value of function f during the i'th regeneration cycle. For all $i \in \{1, \ldots, M\}$, Y_i are *i. i. d. r. v.* and [37]:

$$\alpha = \frac{E[Y_i]}{E[d_i]}.$$

This means that the expected value of the function of the steady state equals the expected accumulated value of the function on the regeneration cycle, averaged by the expected length of the regeneration cycle. This suggests the *p. e.*

$$A = \frac{\sum_{i=1}^{M} Y_i}{\sum_{i=1}^{M} d_i} = \frac{\overline{Y}}{\overline{d}}, \text{ where } \overline{Y} = \frac{1}{M}\sum_{i=1}^{M} Y_i \text{ and } \overline{d} = \frac{1}{M}\sum_{i=1}^{M} d_i. \tag{5.11}$$

This estimate is *consistent*, but *biased* [101]. There are other, less *biased*, estimates such as the *'jackknife estimate'* [101], but we do not consider them here because of their complex structure.

Example 15 *For the bank model from Example 8, one of the measures of interest is* $\alpha_M = E[I_M(S_\infty)]$*, i. e. the probability of having* C_{max} *customers in the bank at any particular work hour. The DTMC model of the bank, given in Example 10, is irreducible and has finitely many states, thus the regenerative method can be applied. Let us take the state* 0 *to be a regeneration point, as in Example 13. Let us process* $M = 5$ *regeneration cycles as shown in Figure 5.6. States* $\{2, 5, 8\}$ *are the ones where* $S_\infty = C_{max}$*. The observed pairs* (Y_i, d_i) *are listed in Table 5.1.*

The p. e. *A then is computed as follows:*

$$A = \frac{1+2+0+3+1}{3+9+3+9+6} = \frac{7}{30} \approx 0.23.$$

The true value of α_M *is approximately* 0.4*, as computed by MRMC [84].*

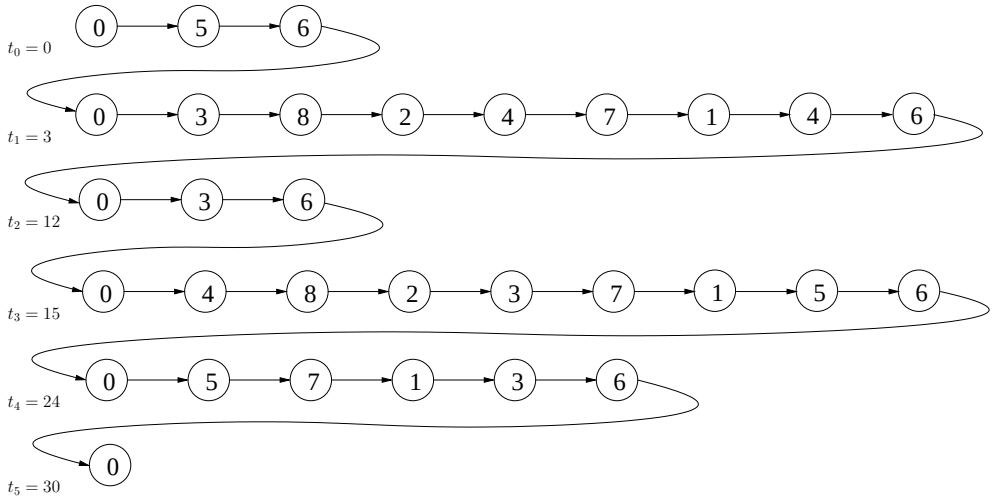


Figure 5.6: Regeneration cycles of the bank example, DTMC

The *c. i.* for the estimate A is:

$$Prob\left(A - \frac{\widetilde{z_s}(\beta) \cdot V_A}{\overline{d} \cdot \sqrt{M}} \leq \alpha \leq A + \frac{\widetilde{z_s}(\beta) \cdot V_A}{\overline{d} \cdot \sqrt{M}}\right) \approx 1 - \beta, \tag{5.12}$$

where $V_A^2 = V_Y^2 - 2 \cdot A \cdot V_{Y,d} + A^2 \cdot V_d^2$ and:

$$V_Y^2 = \frac{1}{M-1}\sum_{i=1}^{M}(Y_i - \overline{Y})^2,\ V_d^2 = \frac{1}{M-1}\sum_{i=1}^{M}(d_i - \overline{d})^2,\ V_{Y,d} = \frac{1}{M-1}\sum_{i=1}^{M}(Y_i - \overline{Y})(d_i - \overline{d}).$$

The *c. i.* (5.12) is based on the *r. v.*

$$\frac{\overline{d} \cdot \sqrt{M} \cdot (A - \alpha)}{V_A} = \frac{\sum_{i=0}^{M}(Y_i - \alpha d_i)}{V_A/\sqrt{M}}, \tag{5.13}$$

which has approximately the Student's t_{M-1} distribution[5]. This justifies why in Equation (5.12) we use $\widetilde{z_s}(\beta)$, as in Equation (5.9).

Example 16 *Let us compute the* 95% c. i. *for the* p. e. A *of Example 15. It is easy to compute that* $\overline{d} = 6$, $\overline{Y} = \frac{7}{5}$, $A = \frac{7}{30}$ *and thus* $V_Y^2 = \frac{13}{10}$, $V_d^2 = 9$, $V_{Y,d}^2 = 3$, *which makes*

$$V_A = \sqrt{\frac{13}{10} - 2 \cdot \frac{7}{30} \cdot 3 + \frac{49}{900} \cdot 9} = \frac{\sqrt{39}}{10}.$$

For the 0.95 *confidence,* β *equals* 0.05. *The Student's* t_4 *distribution yields that:*

$$Prob\left(t_4 \leq \widetilde{z_s}(0.05)\right) = 0.975$$

[5]Due to Lemma 17 with $X_i = Y_i - \alpha d_i$. Note that $E[X_i] = 0$, and X_i turns out to be approximately normally distributed because of the way Y_i and d_i are defined.

is satisfied for $\widetilde{z}_s\,(0.05) = 2.77645$. *Now the* c. i. *can be derived as:*

$$\left[\frac{7}{30} - \frac{2.77645 \cdot \frac{\sqrt{39}}{10}}{6 \cdot \sqrt{5}},\ \frac{7}{30} + \frac{2.77645 \cdot \frac{\sqrt{39}}{10}}{6 \cdot \sqrt{5}}\right],$$

from which we finally obtain that with 95% *confidence the value of* α_M *belongs to the interval* [0.181638648, 0.285028019].

Recall that α_M *is approximately* 0.4 *and thus the derived* c. i. *is not correct. The latter highlights the probabilistic aspect of* c. i., *as for the given example we have* 5% *chance of deriving a wrong interval. Nevertheless, the* c. i. *approach is sound because deriving, e. g.,* 100 *intervals for* α_M *we can expect to have up to* 5 *of them to be wrong.*

5.6 Discrete-time method for simulating CTMCs

In the paper of A. Hordijk et al. [70], a method for simulating CTMCs using discrete event simulation is described. The method is aimed at estimating the steady-state measures. It is based on the fact that the continuity of time influences the point estimate and the confidence interval only by the exit rates of the CTMC. Therefore, the authors suggest to perform simulations on the embedded DTMC using the regenerative method that is explained in the previous section. The *p. e.* and the *c. i.* are derived from specially constructed observations, induced by the observations of the embedded DTMC. The results of the paper, which will be used in the sequel, are as follows.

Consider an irreducible CTMC $\{X_t \mid t \in \mathbb{R}_{\geq 0}\}$, with a finite state space S and generator matrix $\mathbf{Q} = (q_{i,j})_{i,j \in S}$. Assume for function $f : S \to \mathbb{R}$, the value $\alpha = E\left[f\left(X\right)\right]$ has to be estimated where $\lim\limits_{t \to \infty} (X_t) = X$.

For a generator matrix $\mathbf{Q}$ construct a probability matrix $\mathbf{P}$ that defines an embedded DTMC with $\mathbf{P}_n$ being the *r. v.* indicating the state of the DTMC at the n'th epoch.

In order to apply the regenerative method for the embedded DTMC, select a fixed state $s_0 \in S$ and define regeneration points $t_0, \ldots, t_M$ based on returning to the state s_0. Regeneration simulation on the embedded DTMC then results in the sets of observations $\left\{\mathbf{P}_{t_{(i-1)}}, \ldots, \mathbf{P}_{t_i - 1}\right\}$ for each regeneration cycle $i \in 1, \ldots, M$ and provides the basis for the following observations:

$$S_i = \sum_{n=t_{(i-1)}}^{t_i - 1} \frac{f\left(\mathbf{P}_n\right)}{q_{\mathbf{P}_n}} \qquad \text{and} \qquad T_i = \sum_{n=t_{(i-1)}}^{t_i - 1} \frac{1}{q_{\mathbf{P}_n}} \tag{5.14}$$

where $q_{\mathbf{P}_n} = -q_{\mathbf{P}_n, \mathbf{P}_n}$ is the exit rate of the state $\mathbf{P}_n$ in the CTMC. The intuition behind this construction is the following: $\frac{1}{q_{\mathbf{P}_n}}$ is the expected time spent in the state $\mathbf{P}_n$ of the CTMC; $\frac{f(\mathbf{P}_n)}{q_{\mathbf{P}_n}}$ is the expected accumulated value of the function f in this state. T_i is thus the expected time the CTMC spends in the i'th regeneration cycle, and S_i is the expected accumulated value of the function f on this cycle. This is similar to the definition of Y_i in Equation (5.10).

A point estimate A' of α is then given as:

$$A' = \frac{\overline{S}}{\overline{T}}, \text{ where } \overline{S} = \frac{1}{M} \sum_{i=1}^{M} S_i \text{ and } \overline{T} = \frac{1}{M} \sum_{i=1}^{M} T_i,$$

like the *p. e.* of the regenerative method, see Equation (5.11).

Example 17 *In Example 15, the* p. e. *for* α_M *is obtained with the bank modeled as a DTMC. Let us consider a CTMC model for the same property, by taking* $\lambda = 1$, $\beta = 3$, *and* $\gamma = 6$. *The embedded DTMC for this model is exactly the DTMC of Example 16, therefore we can reuse the regeneration cycles simulated earlier. The exit rate for the states* $\{0, 1, 2, 3, 4, 5\}$ *is* 10, *the states* $6, 7, 8$ *have the exit rate* 1. *To derive the* p. e. *and* c. i. *for* α_M, *we should first obtain observations from the simulated regeneration cycles of Figure 5.6, using Equation (5.14):*

$$\begin{array}{ll} S_1 = 1/10 = 0.1, & T_1 = 2 \cdot 1/10 + 1/1 = 1.2 \\ S_2 = 1/1 + 1/10 = 1.1, & T_2 = 6 \cdot 1/10 + 3 \cdot 1/1 = 3.6 \\ S_3 = 0.0, & T_3 = 2 \cdot 1/10 + 1/1 = 1.2 \\ S_4 = 1/1 + 2 \cdot 1/10 = 1.2, & T_4 = 6 \cdot 1/10 + 3 \cdot 1/1 = 3.6 \\ S_5 = 1/10 = 0.1, & T_5 = 4 \cdot 1/10 + 2 \cdot 1/1 = 2.4. \end{array}$$

The p. e. A' *then is computed as follows:*

$$A' = \frac{0.1 + 1.1 + 0.0 + 1.2 + 0.1}{1.2 + 3.6 + 1.2 + 3.6 + 2.4} = \frac{5}{24} \approx 0.208.$$

Assuming that the expected value of the function of the *r. v.* X is finite, i. e.:

$$\sum_{i \in S} \mid f\,(i) \mid \cdot Prob\,(X = i) < \infty,$$

the confidence interval for the estimate A' is given as:

$$Prob\left(A' - \frac{\widetilde{z_s}\,(\beta) \cdot V_{A'}}{\overline{T} \cdot \sqrt{M}} \leq \alpha \leq A' + \frac{\widetilde{z_s}\,(\beta) \cdot V_{A'}}{\overline{T} \cdot \sqrt{M}}\right) \approx 1 - \beta \tag{5.15}$$

where $V_{A'}^2 = V_S^2 - 2 \cdot A' \cdot V_{S,T} + (A')^2 \cdot V_T^2$, and:

$$V_S^2 = \frac{1}{M-1}\sum_{i=1}^{M}(S_i - \overline{S})^2,\ V_T^2 = \frac{1}{M-1}\sum_{i=1}^{M}(T_i - \overline{T})^2,\ V_{S,T} = \frac{1}{M-1}\sum_{i=1}^{M}(S_i - \overline{S})(T_i - \overline{T}).$$

The confidence interval (5.15) is based on the *r. v.*

$$\frac{\overline{T} \cdot \sqrt{M} \cdot (A' - \alpha)}{V_{A'}}$$

which, similarly to the *r. v.* from Equation (5.13), has the Student's t_{M-1} distribution.

Example 18 *Let us compute the* 95% *confidence interval for the* p. e. A' *of Example 17. It is easy to compute that* $\overline{T} = 2.4$, $\overline{S} = 0.5$, $A' = \frac{5}{24}$ *and thus* $V_S^2 = 0.355$, $V_T^2 = 1.44$, $V_{S,T}^2 = 0.66$. *This yields:*

$$V_{A'} = \sqrt{0.355 - 2 \cdot \frac{5}{24} \cdot 0.66 + \frac{25}{576} \cdot 1.44} = \frac{\sqrt{57}}{20}.$$

As in Example 16 we take $\widetilde{z_s}(0.05) = 2.77645$. *Now the* c. i. *can be computed as:*

$$\left[\frac{5}{24} - \frac{2.77645 \cdot \frac{\sqrt{57}}{20}}{2.4 \cdot \sqrt{5}},\ \frac{5}{24} + \frac{2.77645 \cdot \frac{\sqrt{57}}{20}}{2.4 \cdot \sqrt{5}}\right],$$

from which it follows that with approximately 95% *confidence the value of* α_M *belongs to the interval* [0.013033872, 0.403632794]. *Unlike in Example 16, this confidence interval is correct because* $\alpha_M \approx 0.4$. *This can be explained by the fact that the confidence interval width and the point estimate are affected by the exit rates of the CTMC.*

5.7 Bernoulli trials

In this section we introduce Bernoulli trials, discuss the *c. i.* for the mean value of a Bernoulli distributed *r. v.*, and present the strong law of large numbers for Bernoulli trials. These results are going to play an essential role in model checking CSL by discrete event simulation that is discussed in the remainder of this chapter.

Consider a Bernoulli trial with p being the probability of success (S) and $1-p$ the probability of failure (F). The probability space for such an experiment can be defined similarly to how it was done in Example 9. Let *r. v.* X^b be the following indicator function:

$$X^b(\omega) = \begin{cases} 1 & \text{if } \omega = S \\ 0 & \text{if } \omega = F \end{cases}$$

For an experiment consisting of an infinite sequence of independent Bernoulli trials, the sample space Ω contains infinite sequences of single trial outcomes. The sigma field $\mathcal{F}$ on Ω is induced by the sets of finite prefixes, i.e.

$$\mathcal{F} = \{\mathcal{F}_{n,C} | n \in \mathbb{N}, \quad C \subseteq \{S,F\}^n\}, \quad \text{where}$$
$$\mathcal{F}_{n,C} = \{\omega \in \Omega | C \text{ is satisfied on the first } n \text{ elements of } \omega\}$$

which provides a way to define the probability measure in a natural way. Let *r. v.* $\Gamma_M^S : \Omega \to \mathbb{R}$, be $\Gamma_M^S = \sum_{i=1}^{M} X_i^b$ counting the number of successes in the first M experiments of a trial sequence $\omega \in \Omega$. For example, $\omega_1 = (S, F, F, F, \ldots, S, S, F, \ldots)$ is an element of Ω, and $\Gamma_4^S(\omega_1) = X^b(S) + X^b(F) + X^b(F) + X^b(F) = 1$.

The *c. i.* for the mean value of a Bernoulli distributed *r. v.* The *r. v.* X^b is known to have the following values of mean and variance:

$$E\left[X^b\right] = p, \qquad Var\left[X^b\right] = p \cdot (1-p). \tag{5.16}$$

On the basis of this knowledge and the Central Limit Theorem 15, taking $\overline{X} = \Gamma_M^S / M$, one can derive the so called standard *Wald c. i.* for $E\left[X^b\right]$:

$$Prob\left(\overline{X} - \frac{\widetilde{z_n}(\beta) \cdot \sqrt{\overline{X} \cdot (1-\overline{X})}}{\sqrt{M}} \le p \le \overline{X} + \frac{\widetilde{z_n}(\beta) \cdot \sqrt{\overline{X} \cdot (1-\overline{X})}}{\sqrt{M}}\right) \approx 1 - \beta. \tag{5.17}$$

Note that this *c. i.* is different from the *c. i.* given by Equation (5.8). The former employs the variance given by Equation (5.16), where p is substituted with the sample mean $\overline{X}$, whereas the latter interval uses the standard sample variance as provided by Equation (5.7).

In [22] the *c. i.* given by Equation 5.17 is shown to have chaotic coverage properties. Furthermore, the common textbook prescriptions regarding its safety are indicated to be misleading and not trustworthy. One of the suggested substitutes for the Wald *c. i.* is the *Agresti-Coull c. i.* given by:

$$Prob\left(\widetilde{X}-\frac{\widetilde{z_n}(\beta)\cdot\widetilde{V}}{\sqrt{M+\left(\widetilde{z_n}(\beta)\right)^2}}\le p\le\widetilde{X}+\frac{\widetilde{z_n}(\beta)\cdot\widetilde{V}}{\sqrt{M+\left(\widetilde{z_n}(\beta)\right)^2}}\right)\approx 1-\beta \tag{5.18}$$

$$\text{where}\qquad \widetilde{X}=\frac{\Gamma_M^S+0.5\cdot\left(\widetilde{z_n}(\beta)\right)^2}{M+\left(\widetilde{z_n}(\beta)\right)^2}\qquad\text{and}\qquad \widetilde{V}=\sqrt{\widetilde{X}\cdot\left(1-\widetilde{X}\right)}.$$

This interval possesses good coverage properties for all values of p and is easy to use due to its simple form.

The strong law of large numbers for Bernoulli trials [128]. For the *r. v.* Γ_M^S the strong law of large numbers for Bernoulli trials states that:

$$Prob\left(\lim_{M\to\infty}\left(\frac{\Gamma_M^S}{M}\right)=p\right)=1, \tag{5.19}$$

which is often referred as an almost-sure (*a. s.*) convergence and is denoted by $\frac{\Gamma_M^S}{M}\xrightarrow[M\to\infty]{a.\ s.} p$. Put in a nutshell, Equation (5.19) states that the probability of the event:

$$\left\{\omega\in\Omega|\lim_{M\to\infty}\left(\frac{\Gamma_M^S(\omega)}{M}\right)=p\right\},$$

equals one. Note that the limit is checked on the elements of the sample space, and therefore in every particular case we deal with a simple convergence of the numerical sequence.

Chapter 6

Model checking CSL

In this chapter we present an application of discrete event simulation to model checking of probabilistic systems. More specifically, we concentrate on model checking CSL properties by simulation of finite-state CTMCs. Being statistical in nature, such an approach cannot guarantee that the verification result is 100% correct. Yet, it allows to bound the probability of generating an incorrect answer to a verification problem, and, unlike the numerical approaches[1], model checking using simulations does not suffer from the state-space explosion.

Techniques for model checking CSL (PCTL) properties using simulations have already been developed. For example in [146], later extended by [144], an algorithm based on Monte Carlo simulation and hypothesis testing for non-explosive stochastic discrete-event systems is suggested. In [121], the algorithms of [146] are extended to statistically verify black-box, deployed systems with a passive observer. Both statistical approaches [146, 121] considered a sub-logic of CSL that excludes steady-state and unbounded-reachability properties. In [139], the algorithm is extended to deal with a subclass of unbounded-reachability problems. In [122] the statistical verification method of [146] is extended to verify unbounded-reachability properties of CSL (or PCTL) on finite-state CTMCs (DTMCs), and SMCs. All these approaches presume an "on-the-fly" model generation.

Contrary to the above mentioned techniques, our approach is based on Monte Carlo simulation and derivation of confidence intervals. We provide statistical algorithms for model checking the most interesting CSL (PCTL) operators, such as steady-state, unbounded-reachability, and time-interval reachability operators. In addition, when model checking unbounded-reachability or steady-state properties of CSL, we do simulations on the embedded DTMC. The latter simplifies simulation runs and also lets the corresponding techniques for model checking of PCTL properties on DTMCs to be easily derived. In this work we do not consider nested formulas, and working with finite-state systems, we assume that we can deduce the structure of the Markov chain. For instance we can detect bottom strongly connected components of the Markov chain.

Note that, in this chapter we rely on the preliminary material introduced in Chapters 1 and 5. Also, the comparative experimental study of the model-checking techniques derived below and the ones based on hypothesis testing is provided in Chapter 7.

[1] Numerical model checking is carried out by symbolic and numerical methods.

The remainder of this chapter is organized as follows. In Section 6.1 we explain the way of applying confidence intervals to model-checking probabilistic formulas of CSL. Similar to numerical model checking, the procedure falls into two part: *(i)* deriving a *c. i.* for the probability value; *(ii)* checking the *c. i.* against the probability constraint provided by the formula. Section 6.2 deals with the unbounded-reachability operator, using confidence intervals and terminating simulations. Then this technique, in combination with the approach of A. Hordijk et al., is applied to model checking of steady-state properties in Section 6.3. Later, in Section 6.4, we discuss an application of terminating simulation and *c. i.* to model checking of time-bounded reachability properties.

6.1 Confidence intervals and model checking

Let us consider the verification of the three most important operators of CSL: the unbounded-until operator $\mathrm{P}_{\bowtie b}(\mathcal{A}\,\mathrm{U}\,\mathcal{G})$, the steady-state operator $\mathrm{S}_{\bowtie b}(\mathcal{G})$, and the time-interval until operator $\mathrm{P}_{\bowtie b}\left(\mathcal{A}\,\mathrm{U}^{[t_1,t_2]}\,\mathcal{G}\right)$, with $t_1, t_2 \in \mathbb{R}_{\geq 0}$ and $t_1 \leq t_2$. We assume that $\bowtie \in \{<, \leq, >, \geq\}$ and, since we do not consider nested formulas, both $\mathcal{A}$ and $\mathcal{G}$ are treated as sets of states, referred to as *allowed states* and *goal states*, respectively.

In order to verify the formulas $\mathrm{P}_{\bowtie b}(\mathcal{A}\,\mathrm{U}\,\mathcal{G})$, $\mathrm{P}_{\bowtie b}\left(\mathcal{A}\,\mathrm{U}^{[t_1,t_2]}\,\mathcal{G}\right)$ or $\mathrm{S}_{\bowtie b}(\mathcal{G})$, we intend to apply the following procedure. First, for every initial state $s_0 \in S$ the probability $\widetilde{p}$ ($= \mathit{Prob}(s_0,\, \mathcal{A}\,\mathrm{U}\,\mathcal{G})$, $= \mathit{Prob}\left(s_0,\, \mathcal{A}\,\mathrm{U}^{[t_1,t_2]}\,\mathcal{G}\right)$ or $= \mathit{Prob}^{\infty}(s_0,\, \mathcal{G})$) is estimated in a form of the *c. i.* Second, the *c. i.* of $\widetilde{p}$ is checked against the probability constraint $\bowtie b$, to assess whether s_0 satisfies the given formula or not.

Leaving the task of computing the *c. i.* of $\widetilde{p}$ for later, further we concentrate on the second step of the outlined approach. There are two important reasons for that. First, this procedure should be universal for all considered operators. Second, because of the probabilistic nature of the *c. i.*, the procedure should guarantee the correctness of the result with some (predefined) confidence. The latter will imply certain constraints on the way the *c. i.* of $\widetilde{p}$ has to be derived.

The remainder of this section is organized in the following way. We begin with Section 6.1.1 that discusses a general idea of comparing the *c. i.* against the probability constraint. In this section we derive a condition that the *c. i.* has to satisfy in order to guarantee a predefined confidence of the procedure's result. Further, in Section 6.1.2 we present a complete algorithm for comparing the *c. i.* against the probability constraint. Section 6.1.3 briefly mentions the differences between the suggested algorithm and the approach based on hypothesis testing.

6.1.1 Confidence of model checking results

For simplicity, instead of the *c. i.* let us first consider two bounds $A_l,\ A_r \in \mathbb{R}_{\geq 0}$ such that we know that $A_l \leq \widetilde{p} \leq A_r$. Since the value of $\widetilde{p}$ is unknown, assessing whether $\widetilde{p} \bowtie b$ holds can be done based on the bounds A_l and A_r in a straightforward manner. Clearly, such an assessment, for all allowed $\bowtie$, is possible only if $b \notin [A_l,\, A_r]$ and thus the check yields three possible answers: positive (*TRUE*), negative (*FALSE*), or "Don't know" (*NN*).

Let us extend the reasoning above to the situation when the bounds for $\widetilde{p}$ are provided in the form of the *c. i.* Then, for a given confidence ξ and sample size $M \in \mathbb{N}_{\geq 2}$, we have:

$$Prob\left(A_l\left(\overrightarrow{\mathbf{X}}\right) \leq \widetilde{p} \leq A_r\left(\overrightarrow{\mathbf{X}}\right)\right) \approx \xi. \tag{6.1}$$

Equation (6.1) guarantees that the sampled intervals $\left[A_l\left(\overrightarrow{\mathbf{X}}\right),\ A_r\left(\overrightarrow{\mathbf{X}}\right)\right]$ contain $\widetilde{p}$ in about $100 \cdot \xi\,\%$ of the cases. Note that, in this work we do not dwell on the quality of the *c. i.* given by Equation (6.1). In other words, we assume that the sample size M is large enough[2] to provide an accuracy that allows not to influence our model-checking techniques.

Now, let us concentrate on two distinct problems that arise when we attempt to use the *c. i.* of $\widetilde{p}$ in order to decide whether or not $\widetilde{p} \bowtie b$ holds:

- Like for the fixed bounds A_l and A_r, if the sampled *c. i.* contains b then the solution to the model-checking problem is unknown. Thus, similar to model checking by means of hypothesis testing [144, 121, 122], the analysis based on the *c. i.* is inconclusive if $b = \widetilde{p}$. Clearly, in this case with probability ξ we sample a correct *c. i.* that contains both $\widetilde{p}$ and b.

- Due to the probabilistic nature of the *c. i.*, the result of the comparison between the *c. i.* and constraint $\bowtie\ b$ becomes probabilistic itself. This means that, in order to give a correct answer to $\widetilde{p} \bowtie b$, it is not enough to check the *c. i.* of $\widetilde{p}$ against $\bowtie\ b$. In addition, we have to provide a confidence with which the result of such comparison provides a correct answer to $\widetilde{p} \bowtie b$.

To avoid the first problem, from now on we will assume that we only consider values of b such that $|b - \widetilde{p}| = \delta$ with $\delta \in \mathbb{R}_{>0}$. The solution to the second problem is not so straightforward and requires a deeper understanding of the situation.

The borders of the sampled *c. i.* can vary from simulation to simulation. The latter implies that the confidence of having a correct answer to $\widetilde{p} \bowtie b$ may be unknown, even if we know with what probability every other sampled interval contains $\widetilde{p}$. Consider any fixed confidence ξ and a sample size M then we can always have b being chosen so close to $\widetilde{p}$ that in many cases the sampled *c. i.* will contain b. Then the % of *NN* answers for $\widetilde{p} \bowtie b$ is going to be high (up to $100 \cdot \xi\,\%$) making the % of correct definite answers comparable to the % of incorrect definite answers. Fortunately, this problem can be solved by *sequential* confidence intervals.

Note that in this work we do not consider the proper sequential *c. i.* derivation such as discussed in [44, 30]. Instead, we use a naive approach where we simply increase the sample size until the derived *c. i.* becomes narrow enough. The latter can cause the decrease of the confidence levels, although this was not observed in our experiments presented in Chapter 7. In the following we assume that the suggested sequential procedures do not change the confidence levels.

Assume that we have a fixed confidence ξ and a variable sample size M. The width of the *c. i.* depends on M, namely the larger the sample the tighter is the *c. i.* (cf. Section 5.3). If we compute the *c. i.* of $\widetilde{p}$ by increasing M until $A_r\left(\overrightarrow{\mathbf{X}}\right) - A_l\left(\overrightarrow{\mathbf{X}}\right) < \delta$ then we guarantee that the correct *c. i.* does not contain b. This means that at least[3]

[2] For more details see Section 5.3.

[3] An incorrect *c. i.* of $\widetilde{p}$ can still result in the correct answer to $\widetilde{p} \bowtie b$.

in $100 \cdot \xi\,\%$ cases $\widetilde{p} \bowtie b$ will be provided with a definite and correct answer. As a consequence, the combined percentage of incorrect and "Don't know" answers should not exceed $100 \cdot (1 - \xi)$ %.

In the solution above, δ is defined using $\widetilde{p}$ which is unknown. Therefore, we suggest to use a user-defined estimate of δ which we denote as $\delta' \in \mathbb{R}_{>0}$. This value has to be guessed and a good guess is the one for which $\delta' \leq \delta$ holds. The latter cannot be checked exactly but by repeating simulations one can empirically evaluate the quality of the given δ', since for a proper value the combined percentage of incorrect and *NN* answers should not exceed $100 \cdot (1 - \xi)$ %.

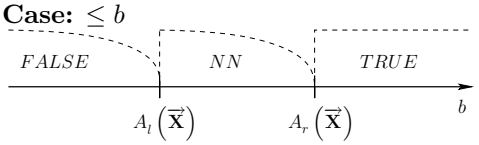
6.1.2 Checking the *c. i.* against the probability constraint

Algorithm 3 *checkBoundVSConfInt* $\left(\bowtie, b, \left[A_l\left(\overrightarrow{\mathbf{X}}\right), A_r\left(\overrightarrow{\mathbf{X}}\right)\right]\right)$

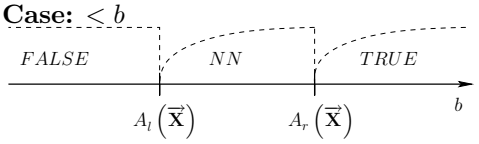
Require: $A_r\left(\overrightarrow{\mathbf{X}}\right) - A_l\left(\overrightarrow{\mathbf{X}}\right) < \delta$

```
1: RESULT := NN
2: if ⋈ ∈ {≤} then /*Case 1*/
3:    if b ≥ A_r(X⃗) then
4:       RESULT := TRUE
5:    else if b < A_l(X⃗) then
6:       RESULT := FALSE
7:    end if
8: else if ⋈ ∈ {<} then /*Case 2*/
9:    if b > A_r(X⃗) then
10:      RESULT := TRUE
11:   else if b ≤ A_l(X⃗) then
12:      RESULT := FALSE
13:   end if
14: else if ⋈ ∈ {≥} then /*Case 3*/
15:   if b ≤ A_l(X⃗) then
16:      RESULT := TRUE
17:   else if b > A_r(X⃗) then
18:      RESULT := FALSE
19:   end if
20: else if ⋈ ∈ {>} then /*Case 4*/
21:   if b < A_l(X⃗) then
22:      RESULT := TRUE
23:   else if b ≥ A_r(X⃗) then
24:      RESULT := FALSE
25:   end if
26: end if
27: return RESULT
```

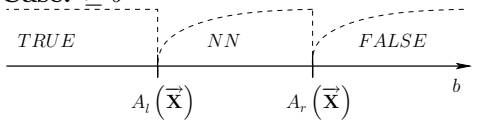
1. **Case:** $\leq b$



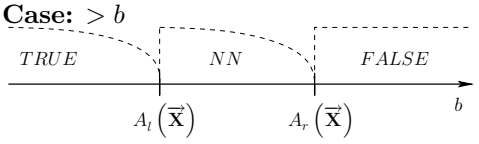
2. **Case:** $< b$



3. **Case:** $\geq b$



4. **Case:** $> b$



Now let us consider Algorithm 3 that checks the *c. i.* of $\widetilde{p}$ against the probability constraint $\bowtie\ b$. This algorithm has three parameters, the binary operator $\bowtie$, the probability bound b, and the *c. i.* of $\widetilde{p}$. The result can be one of the three possible

outcomes *TRUE*, *FALSE*, or *NN*. The precondition $A_r\left(\overrightarrow{\mathbf{X}}\right) - A_l\left(\overrightarrow{\mathbf{X}}\right) < \delta$ ensures that the confidence level of the definite answers given by the algorithm is at least ξ.

Assuming a correct *c. i.*, i. e. $A_l\left(\overrightarrow{\mathbf{X}}\right) \leq \widetilde{p} \leq A_r\left(\overrightarrow{\mathbf{X}}\right)$, consider Algorithm 3 for the case of $\bowtie$ equal to $\leq$. This corresponds to lines 2 through 7. First, on line 3 it is tested whether $A_r\left(\overrightarrow{\mathbf{X}}\right) \leq b$. If this holds then, since we know that $\widetilde{p} \leq A_r\left(\overrightarrow{\mathbf{X}}\right)$, we can conclude that $\widetilde{p} \leq b$ and the algorithm's outcome is *TRUE*. Similarly, if $b < A_l\left(\overrightarrow{\mathbf{X}}\right)$ then we have $\widetilde{p} > b$ and the result is *FALSE*. In the remaining case, i. e. $A_l\left(\overrightarrow{\mathbf{X}}\right) \leq b < A_r\left(\overrightarrow{\mathbf{X}}\right)$ we cannot provide any definite answer and the algorithm returns *NN*.

The illustrations along the algorithm correspond to the *if*-cases marked in the pseudo code by comments. These pictures provide an extra insight into the algorithm's decision process, i. e. when this or that value is returned depending on the binary operator $\bowtie$ and the position of b with respect to the *c. i.* borders. The dash lines denote intervals for b in which Algorithm 3 returns a particular value. Here we use a cornered line to denote inclusion of $A_l\left(\overrightarrow{\mathbf{X}}\right)$ or $A_r\left(\overrightarrow{\mathbf{X}}\right)$ into the interval, and rounded line to denote the exclusion.

At this point, the way of comparing the sampled *c. i.* of $\widetilde{p}$ to a probability constraint and the necessity of ensuring $A_r\left(\overrightarrow{\mathbf{X}}\right) - A_l\left(\overrightarrow{\mathbf{X}}\right) < \delta$ for the proper confidence levels are clear. Further, we briefly mention the differences between Algorithm 3 and the approach based on hypothesis testing. The remainder of this chapter will be devoted to deriving the *c. i.* for probabilities $Prob\left(s_0,\ \mathcal{A}\ \mathrm{U}\ \mathcal{G}\right)$, $Prob\left(s_0,\ \mathcal{A}\ \mathrm{U}^{[t_1,t_2]}\ \mathcal{G}\right)$ and $Prob^{\infty}\left(s_0,\ \mathcal{G}\right)$, and specifying the complete model checking procedures.

6.1.3 Confidence intervals and hypothesis testing

Let us note that the algorithm presented in Section 6.1.2 implements criteria different from the acceptance criteria used in model checking by hypothesis testing [144, 122]. As a result we have two main differences: *(i)* our approach allows for the indefinite answer (*NN*); *(ii)* we do not have an explicit notion of the indifference region.

The first difference might not allow for a straightforward way of handling nested properties. Nevertheless, the use of undecided results is sound and has been suggested for numerical model checking in [60] and for hypothesis testing in Section 5 of [142].

The second difference, see Section 7.1, requires us to use *c. i.* of the width $< \delta$, whereas – under the same conditions – in hypothesis testing we would have to use the indifference region of the width less than only $2 \cdot \delta$. This can cause our model checking algorithms to require more samples than needed for the ones based on hypothesis testing. An alternative approach to comparing the *c. i.* of $\widetilde{p}$ against the probability constraint $\bowtie b$ was discussed in Section 4.2 of [142]. That procedure does not allow for an indefinite answer but was shown to require the *c. i.* of the width up to $2 \cdot \delta$.

6.2 Unbounded-until operator

Let us first recall the numerical model checking of $\mathrm{P}_{\bowtie b}\left(\mathcal{A}\ \mathrm{U}\ \mathcal{G}\right)$ on a CTMC $(S,\ \mathbf{Q},\ L)$. It consists of computing the probabilities $Prob\left(s_0,\ \mathcal{A}\ \mathrm{U}\ \mathcal{G}\right)$ for all states $s_0 \in S$, and

then selecting the states for which the probability bound $\bowtie b$ is satisfied. More precisely, as described in Section 1.2.2, the model-checking procedure typically looks as follows:

1. States $\mathcal{G}$ and $\mathcal{I} = S \setminus (\mathcal{A} \cup \mathcal{G})$ are made absorbing, resulting in a new generator matrix $\mathbf{Q}\,[\mathcal{I} \cup \mathcal{G}]$.
2. An extra step may be taken to make $B_{\mathcal{A},\mathcal{G}} = \{\text{BSCCs in } \mathcal{A} \setminus \mathcal{G}\}$ states absorbing. This is done because the unbounded until formula is not satisfiable in states from $B_{\mathcal{A},\mathcal{G}}$. This step results in the generator matrix $\mathbf{Q}^B$.
3. Since time is of no importance, the embedded DTMC is considered; its probability matrix is denoted $\mathbf{P}^B$.
4. A system of linear equations is solved to obtain $\mathit{Prob}\,(s_0,\ \mathcal{A}\ \mathrm{U}\ \mathcal{G})$ for all states $s_0 \in S$ at once.
5. Finally, states $s_0 \in S$ are selected such that $\mathit{Prob}\,(s_0,\ \mathcal{A}\ \mathrm{U}\ \mathcal{G}) \bowtie b$ holds.

When using simulations, we utilize the first three steps of the above mentioned procedure. Then two important remarks are in order. First, in the DTMC represented by the matrix $\mathbf{P}^B$ the state space is divided into three disjoint parts: the "allowed" transient states $\mathcal{A} \setminus (\mathcal{G} \cup B_{\mathcal{A},\mathcal{G}})$, the "bad states" $B_{\mathcal{A},\mathcal{G}} \cup \mathcal{I}$, and the "goal states" $\mathcal{G}$. The latter two sets of states are absorbing. Second, because we are only interested in the probability of reaching a $\mathcal{G}$ state in the long run, we can safely discard all the self-loops, if any, of the transient states in the embedded DTMC $\mathbf{P}^B$. In case the self loop of a state $s \in \mathcal{A} \setminus (\mathcal{G} \cup B_{\mathcal{A},\mathcal{G}})$ is removed, the probabilities on the remaining outgoing transitions of s should be renormalized in order to form a proper distribution. Due to the first remark, we conclude that the DTMC represented by $\mathbf{P}^B$ is absorbing. The second one provides us with a way to optimize simulation runs, as excluding the self-loops on transient states will reduce the length of simulation runs.

When estimating $\mathit{Prob}\,(s_0,\ \mathcal{A}\ \mathrm{U}\ \mathcal{G})$ using simulations, we are interested in the long-run measure, namely an estimate of the probability to be in some $\mathcal{G}$ state in the long run, when starting in $s_0 \in \mathcal{A} \setminus (\mathcal{G} \cup B_{\mathcal{A},\mathcal{G}})$. Long-run simulation of an absorbing Markov chain, doing terminating simulations until absorption, is not always practical, because in case of a slowly convergent Markov chain such simulation may take an arbitrary long time. Instead, we first bound the probability $\mathit{Prob}\,(s_0,\ \mathcal{A}\ \mathrm{U}\ \mathcal{G})$ by transient probabilities. Then we use terminating simulation until a time stamp $N \in \mathbb{N}$ in order to provide the *c. i.* for the probabilities bounding $\mathit{Prob}\,(s_0,\ \mathcal{A}\ \mathrm{U}\ \mathcal{G})$. In the end, we use these *c. i.* to derive a *c. i.* for $\mathit{Prob}\,(s_0,\ \mathcal{A}\ \mathrm{U}\ \mathcal{G})$.

All our simulation runs start in state s_0 and thus the probability measure we use is conditional. We denote a state of the discrete time process corresponding to the embedded DTMC $\mathbf{P}^B$ as $\{\mathbf{P}_N \mid N \in \mathbb{N}\}$, where $\mathbf{P}_N$ is the *r. v.* indicating the state of the DTMC at the N'th epoch. Sampling the *r. v.* $\mathbf{P}_N$ we form a sample $\overrightarrow{\mathbf{P}_N} = \left(\mathbf{P}_N^1, \ldots, \mathbf{P}_N^M\right)$ of $M \in \mathbb{N}_{\geq 2}$ independent observations.

The rest of this section is organized as follows. In Section 6.2.1 we show how the probability $\mathit{Prob}\,(s_0,\ \mathcal{A}\ \mathrm{U}\ \mathcal{G})$ can be bounded (from above and below) by transient probabilities. The latter turn out to be the mean values of Bernoulli-distributed

random variables. Then, in Section 6.2.2 we show how these probabilities can be estimated using both standard and Agresti-Coull confidence intervals. Since the *c. i.* of $Prob\,(s_0,\ \mathcal{A}\ \mathrm{U}\ \mathcal{G})$ is derived as a composition of the *c. i.* for transient probabilities, in Section 6.2.2 we introduce a notion of dependent *c. i.* This notion is used in Section 6.2.3 that is devoted to deriving the *c. i.* of $Prob\,(s_0,\ \mathcal{A}\ \mathrm{U}\ \mathcal{G})$. As we will see, there are several *c. i.* available for $Prob\,(s_0,\ \mathcal{A}\ \mathrm{U}\ \mathcal{G})$. In order to ensure a desired level of confidence, these intervals are based on different numbers of samples. This allows to choose the best between the available *c. i.* of $Prob\,(s_0,\ \mathcal{A}\ \mathrm{U}\ \mathcal{G})$ and this is done in Section 6.2.4. Further, in Section 6.2.5 we analyze the behavior of the chosen *c. i.* with respect to its parameters, such as the sample size and the depth of terminating simulation. This analysis helps to devise an efficient model-checking algorithm for $\mathrm{P}_{\bowtie b}\,(\mathcal{A}\ \mathrm{U}\ \mathcal{G})$ that is presented in Section 6.2.6.

6.2.1 Bounding $Prob\,(s_0,\ \mathcal{A}\ \mathrm{U}\ \mathcal{G})$ by transient probabilities

Below, we provide a way of bounding the probability $Prob\,(s_0,\ \mathcal{A}\ \mathrm{U}\ \mathcal{G})$ with transient probabilities which can be represented as the mean values of Bernoulli-distributed random variables. For that, let us consider the following indicator function with $S' \subseteq S$:

$$I_{S'}\,(s) = \begin{cases} 1 & \text{if } s \in S' \\ 0 & \text{else} \end{cases}$$

For any $N \in \mathbb{N}$ and $k \in \mathcal{N} = \{\{g\}, \{b\}, \{t\}, \{g,t\}, \{b,t\}\}$, let us define the *r. v.* $f_k\,(\mathbf{P}_N)$ and its mean value α_k^N as follows[4]:

$$\begin{array}{ll}
f_g\,(\mathbf{P}_N) = I_{\mathcal{G}}\,(\mathbf{P}_N), & \alpha_g^N = E\left[f_g\,(\mathbf{P}_N)\right], \\
f_t\,(\mathbf{P}_N) = I_{\mathcal{A}\setminus(\mathcal{G}\cup B_{\mathcal{A},\mathcal{G}})}\,(\mathbf{P}_N), & \alpha_t^N = E\left[f_t\,(\mathbf{P}_N)\right], \\
f_b\,(\mathbf{P}_N) = I_{B_{\mathcal{A},\mathcal{G}}\cup\mathcal{I}}\,(\mathbf{P}_N), & \alpha_b^N = E\left[f_b\,(\mathbf{P}_N)\right], \\
f_{g,t}\,(\mathbf{P}_N) = I_{(\mathcal{A}\setminus B_{\mathcal{A},\mathcal{G}})\cup\mathcal{G}}\,(\mathbf{P}_N), & \alpha_{g,t}^N = E\left[f_{g,t}\,(\mathbf{P}_N)\right], \\
f_{b,t}\,(\mathbf{P}_N) = I_{S\setminus\mathcal{G}}\,(\mathbf{P}_N), & \alpha_{b,t}^N = E\left[f_{b,t}\,(\mathbf{P}_N)\right].
\end{array}$$

Clearly, $f_k\,(\mathbf{P}_N)$ is a Bernoulli-distributed *r. v.* which results in 1 with probability α_k^N and in 0 with probability $1-\alpha_k^N$. Notice that the mean values above define probabilities to be at epoch N in: α_g^N – a goal state, α_t^N – a transient state, α_b^N – a bad state, $\alpha_{g,t}^N$ – a goal or a transient state, $\alpha_{b,t}^N$ – a bad or a transient state. Let us define:

$$\lim_{N\to\infty}\left(\alpha_k^N\right) = \alpha_k,$$

then α_g is our measure of interest, because $\alpha_g = Prob\,(s_0,\ \mathcal{A}\ \mathrm{U}\ \mathcal{G})$. The dependency of α_g on s_0 is implicit via the conditional probability measure.

Proposition 18 below provides us with several useful facts, such as that the values α_g^N, α_b^N and α_t^N (α_g and α_b) form a distribution, and that the probability $\alpha_{g,t}^N$ is the sum of probabilities α_g^N and α_t^N, as the goal and transient states are disjoint.

[4]We mostly omit the curly braces in our notation and write, e. g., α_b^N instead of $\alpha_{\{b\}}^N$.

Proposition 18 *For any $N \in \mathbb{N}$:*

$$\alpha_g^N + \alpha_b^N + \alpha_t^N = 1, \quad \alpha_g + \alpha_b = 1,$$
$$\alpha_{g,t}^N = \alpha_g^N + \alpha_t^N, \quad \alpha_{b,t}^N = \alpha_b^N + \alpha_t^N, \quad \alpha_{g,b}^N = \alpha_g^N + \alpha_b^N.$$

Proof Straightforward. □

Proposition 19 bounds the value of α_g by the values of α_k^N (with $k \in \mathcal{N}$). In other words, this proposition gives us a way to bound the long-run probability $Prob\,(s_0,\ \mathcal{A}\ \mathrm{U}\ \mathcal{G})$ by transient probabilities.

Proposition 19 *For any $N \in \mathbb{N}$ the inequality $A_l \le \alpha_g \le A_r$ holds for any:*

$$A_l \in \left\{\alpha_g^N,\ 1 - \left(\alpha_b^N + \alpha_t^N\right),\ 1 - \alpha_{b,t}^N\right\} \quad \textit{and} \quad A_r \in \left\{\alpha_{g,t}^N,\ \alpha_g^N + \alpha_t^N,\ 1 - \alpha_b^N\right\}.$$

Proof See the proof of Proposition 43 from Appendix C.1. □

Notice that for any allowed choice of A_l and A_r, Proposition 19 provides us with exactly the same inequality. This is not a coincidence because due to Proposition 18 we have:

$$\alpha_g^N = 1 - \left(\alpha_b^N + \alpha_t^N\right) = 1 - \alpha_{b,t}^N, \quad \text{and} \quad \alpha_{g,t}^N = \alpha_g^N + \alpha_t^N = 1 - \alpha_b^N.$$

The difference in the power of different A_l and A_r is revealed only when they are used in combination with the *c. i.* of α_k^N, with $k \in \mathcal{N}$. The latter happens due to non-linearity of the sample variance and will be explained later in more detail. Further, Example 19 illustrates the result of the proposition.

Example 19 *Figure 6.1 gives an example DTMC with the values of α_g^N, α_b^N, α_t^N, $\alpha_{g,t}^N$ and $\alpha_{b,t}^N$ for several values of N. The self-loop on the transient state is not eliminated (although it could be) for increasing the illustrative value of this example.*

As it is shown in Figure 6.1, $\alpha_g = \frac{2}{3}$, and thus for $N = 3$, applying Proposition 19 with $A_l = \alpha_g^N$ and $A_r = \alpha_g^N + \alpha_t^N$, we get the following inequality $\frac{21}{32} \le \frac{2}{3} \le \frac{21}{32} + \frac{1}{64}$.

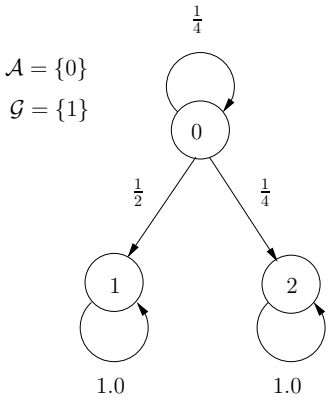
6.2.2 Deriving a *c. i.* of α_k^N

In this section we concentrate on two main things. First, we provide *c. i.* for α_k^N. Second, using the idea of probabilistic events, we define a notion of dependent confidence intervals. The latter is used in the next section where the *c. i.* of α_k^N are employed to produce the *c. i.* of α_g.

For some confidence $1 - \beta$ and a sample $\overrightarrow{\mathbf{P}_N}$ of $M \in \mathbb{N}_{\ge 2}$ independent observations we intend to provide two *c. i.* of α_k^N, based on observation $X_i = f_k\left(\mathbf{P}_N^i\right)$. These *c. i.* will have the form:

$$Prob\left(A_l^k\left(\overrightarrow{\mathbf{P}_N}\right) \le \alpha_k^N \le A_r^k\left(\overrightarrow{\mathbf{P}_N}\right)\right) \approx 1 - \beta, \tag{6.2}$$

and differ by differently defined $A_l^k\left(\overrightarrow{\mathbf{P}_N}\right)$ and $A_r^k\left(\overrightarrow{\mathbf{P}_N}\right)$. Before starting with the intervals, let us introduce the following definition.



N	α_g^N	α_b^N	α_t^N	$\alpha_{g,t}^N$	$\alpha_{b,t}^N$
0	0.0	0.0	1.0	1.0	1.0
1	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{3}{4}$	$\frac{1}{2}$
2	$\frac{5}{8}$	$\frac{5}{16}$	$\frac{1}{16}$	$\frac{11}{16}$	$\frac{3}{8}$
3	$\frac{21}{32}$	$\frac{21}{64}$	$\frac{1}{64}$	$\frac{43}{64}$	$\frac{11}{32}$
...	...	...	...	...	...
∞	$\frac{2}{3}$ [a]	$\frac{1}{3}$	0	$\frac{2}{3}$ [b]	$\frac{1}{3}$

[a] $\lim_{N\to\infty}\left(\alpha_g^N\right) = \alpha_g = \frac{2}{3}$
[b] $\lim_{N\to\infty}\left(\alpha_{g,t}^N\right) = \alpha_g = \frac{2}{3}$

Figure 6.1: Values of α_g^N, α_b^N, α_t^N, $\alpha_{g,t}^N$ and $\alpha_{b,t}^N$ on a simple DTMC, with $s_0 = 0$.

Definition 19 *For any $k \in \mathcal{N}$ and a sample $\overrightarrow{\mathbf{P}_N}$ of size $M \in \mathbb{N}_{\geq 1}$, let $\Gamma_N^k = \Gamma^k\left(\overrightarrow{\mathbf{P}_N}\right) = \sum_{i=1}^{M} f_k\left(\mathbf{P}^i{}_N\right)$.*[5]

Clearly, Γ_N^k defines the number of k-type observations in the sample $\overrightarrow{\mathbf{P}_N}$. For instance, Γ_N^g is the number of goal states in the sample $\overrightarrow{\mathbf{P}_N}$. Below we show that the provided interval borders $A_l^k(.)$ and $A_r^k(.)$ can be seen as functions of Γ_N^k, i.e.

$$A_l^k\left(\overrightarrow{\mathbf{P}_N}\right) = A_l^k\left(\Gamma^k\left(\overrightarrow{\mathbf{P}_N}\right)\right) = A_l^k\left(\Gamma_N^k\right),\ A_r^k\left(\overrightarrow{\mathbf{P}_N}\right) = A_r^k\left(\Gamma^k\left(\overrightarrow{\mathbf{P}_N}\right)\right) = A_r^k\left(\Gamma_N^k\right). \quad (6.3)$$

Note that $A_l^k(.)$ and $A_r^k(.)$, seen as functions of Γ_N^k, will not depend on k at all, nevertheless we prefer to keep this index.

Below, we consider two *c. i.* of α_k^N. The first one is a standard *c. i.* and it will be used for our technical proofs due to its simple structure. The second one is the Agresti-Coull *c. i.* We will use it in the experiments because it is known to provide better accuracy than the standard *c. i.*

The standard confidence interval

In order to provide a *c. i.* for α_k^N we can simply use the general *c. i.* given by Equation (5.8) (cf. Section 5.3):

$$A_l^k\left(\overrightarrow{\mathbf{P}_N}\right) = \overline{X}_k^N - \frac{\widetilde{z_n}(\beta)\cdot\overline{V}_k^N}{\sqrt{M}}, \qquad A_r^k\left(\overrightarrow{\mathbf{P}_N}\right) = \overline{X}_k^N + \frac{\widetilde{z_n}(\beta)\cdot\overline{V}_k^N}{\sqrt{M}}, \quad (6.4)$$

where the following lemma provides us with the compact representation of $\overline{X}_k^N$ and $\overline{V}_k^N$ in terms of Γ_N^k and M.

[5] Note that, unlike in Section 5.7, the lower index (N) of Γ_N^k defines the simulation depth.

Lemma 20 *For a sample* $\overrightarrow{\mathbf{P}_N} = \left(\mathbf{P}_N^1, \ldots, \mathbf{P}_N^M\right)$ *of* $M \in \mathbb{N}_{\geq 1}$ *independent observations,* $X_i = f_k\left(\mathbf{P}_N^i\right)$*, with* $k \in \mathcal{N}$*, and* $\overline{X}_k^N$ *with* $\overline{V}_k^N$ *computed by Equations (5.1) and (5.7) correspondingly, the following holds:*

$$\overline{X}_k^N = \frac{\Gamma_N^k}{M}, \qquad \overline{V}_k^N = \sqrt{\frac{\Gamma_N^k \cdot \left(M - \Gamma_N^k\right)}{M \cdot (M-1)}} = \sqrt{\frac{\overline{X}_k^N \cdot \left(M - \Gamma_N^k\right)}{M-1}}. \tag{6.5}$$

Proof See the proof of Lemma 46 from Appendix C.1. □

The *c. i.* defined by the bounds in Equation (6.4) is rather simple and thus will be used for technical proofs in the subsequent parts of this chapter. The interval, though, is not perfect for practical applications (in our settings) because its general nature does not take into account, e. g., the distribution of the *r. v.* $f_k\left(\mathbf{P}_N\right)$.

The Agresti-Coull confidence interval

As it was noted earlier, $f_k\left(\mathbf{P}_N\right)$ is a Bernoulli-distributed *r. v.* Therefore, we can employ the Agresti-Coull *c. i.* (cf. Equation (5.18) of Section 5.7) specifically designed for the case of Bernoulli trials, for that we should take:

$$\widetilde{A}_l^k\left(\overrightarrow{\mathbf{P}_N}\right) = \widetilde{X}_k^N - \frac{\widetilde{z_n}(\beta) \cdot \widetilde{V}_k^N}{\sqrt{M + \left(\widetilde{z_n}(\beta)\right)^2}}, \qquad \widetilde{A}_r^k\left(\overrightarrow{\mathbf{P}_N}\right) = \widetilde{X}_k^N + \frac{\widetilde{z_n}(\beta) \cdot \widetilde{V}_k^N}{\sqrt{M + \left(\widetilde{z_n}(\beta)\right)^2}} \tag{6.6}$$

$$\text{where} \qquad \widetilde{X}_k^N = \frac{\Gamma_N^k + 0.5 \cdot \left(\widetilde{z_n}(\beta)\right)^2}{M + \left(\widetilde{z_n}(\beta)\right)^2} \qquad \text{and} \qquad \widetilde{V}_k^N = \sqrt{\widetilde{X}_k^N \cdot \left(1 - \widetilde{X}_k^N\right)}.$$

It is important to note that, unless stated otherwise, the results proven in Sections 6.2.3 to 6.2.5 hold for the *c. i.* provided by both Equation (6.4) and (6.6). The latter comes without any proof but is trivial to check.

Symmetric confidence intervals and events

The *c. i.* given by Equation (6.2) can be seen as probabilities of some events. Let us consider the following definition.

Definition 20 *For any* $k \in \mathcal{N}$*, and a sample* $\overrightarrow{\mathbf{P}_N}$ *of* $M \in \mathbb{N}_{\geq 2}$ *independent observations, define the following events:*

$$E^k = A_l^k\left(\overrightarrow{\mathbf{P}_N}\right) \leq \alpha_k^N \leq A_r^k\left(\overrightarrow{\mathbf{P}_N}\right), \quad E_l^k = A_l^k\left(\overrightarrow{\mathbf{P}_N}\right) \leq \alpha_k^N, \quad E_r^k = \alpha_k^N \leq A_r^k\left(\overrightarrow{\mathbf{P}_N}\right).$$

For any $k \in \mathcal{N}$, the *c. i.* of α_k^N defined by Equation (6.2), we have:

$$\mathit{Prob}\left(E^k\right) \approx 1 - \beta, \quad \mathit{Prob}\left(E_l^k\right) \approx 1 - \frac{\beta}{2}, \quad \mathit{Prob}\left(E_r^k\right) \approx 1 - \frac{\beta}{2}, \tag{6.7}$$

where the last two hold due to the use of symmetric *c. i.*, i. e. we have:

$$\mathit{Prob}\left(\alpha_k^N < A_l^k\left(\overrightarrow{\mathbf{P}_N}\right)\right) = \mathit{Prob}\left(A_r^k\left(\overrightarrow{\mathbf{P}_N}\right) < \alpha_k^N\right) \approx 1 - \frac{\beta}{2},$$

due to the way the *c. i.* defined by Equation (5.6) is derived from Equation (5.5).

Since *c. i.* can be seen as probabilities of events we may talk about dependency of these events, i.e. about the dependency of the *c. i.*

Definition 21 *For two* c. i. *of* α_k^N *and* α_l^N *with* $k, l \in \mathcal{N}$*, derived using Equation (6.2), we say that they are dependent* **iff** *the events* E^k *and* E^l *are dependent.*

The notion of dependency between *c. i.* becomes important when deriving the *c. i.* of α_g because the latter involves computation of joint probabilities for several *c. i.*

6.2.3 Deriving *c. i.* of $Prob\,(s_0,\ \mathcal{A}\ \mathrm{U}\ \mathcal{G})$

Our ultimate goal now is to provide a *c. i.* for α_g using the *c. i.* of α_k^N, with $k \in \mathcal{N}$. The latter can be done employing Proposition 19 and Equation (6.2). To be more precise, the *c. i.* of α_g can be deduced as a composition of the *c. i.* for α_k^N. Using the information provided in the previous section, it is easy to see that this composition can be represented as a joint probability of several probabilistic events. We illustrate this idea be means of Example 20, but first we start with the following definition.

Definition 22 *For* $a, b, c \in \mathbb{R}_{[0,1]}$ *if* $a \approx b$ *and* $c \geq a$ *then we write* $c \succeq b$.

Example 20 *Consider the* c. i. *of* α_g^N *and* α_t^N*, and Proposition 19 with* $A_l = \alpha_g^N$ *and* $A_r = \alpha_g^N + \alpha_t^N$*, it is easy to conclude that:*

$$E^g \wedge E_r^t \Longrightarrow A_l^g\left(\overrightarrow{\mathbf{P}_N}\right) \leq \alpha_g \leq A_r^g\left(\overrightarrow{\mathbf{P}_N}\right) + A_r^t\left(\overrightarrow{\mathbf{P}_N}\right). \tag{6.8}$$

The latter implies:

$$Prob\left(A_l^g\left(\overrightarrow{\mathbf{P}_N}\right) \leq \alpha_g \leq A_r^g\left(\overrightarrow{\mathbf{P}_N}\right) + A_r^t\left(\overrightarrow{\mathbf{P}_N}\right)\right) \geq Prob\left(E^g \wedge E_r^t\right) \tag{6.9}$$

where $Prob\,(E^g \wedge E_r^t)$ *is an unknown joint probability, and* $\geq$ *is used, because the probability of the consequent event is always greater or equal than the probability of the antecedent event, see Equation (6.8).*

Clearly, if events E^g and E_r^t (see the example above) are independent then by using Equations (6.7) the joint probability $Prob\,(E^g \wedge E_r^t)$ can be easily computed as follows:

$$Prob\left(E^g \wedge E_r^t\right) = Prob\left(E^g\right) \cdot Prob\left(E_r^t\right) \approx (1-\beta) \cdot \left(1 - \frac{\beta}{2}\right).$$

In the remainder of this section we concentrate on the following three topics. First, we show that the events such as E^g and E_r^t are dependent, if based on the same set of *r. v.* The latter, in case of simulation, is equivalent to using the same sample $\overrightarrow{\mathbf{P}_N}$ for deriving the *c. i.* of α_g^N and α_t^N. Second, we derive several *c. i.* of α_g which, keeping in mind the dependency between the *c. i.* of α_k^N, are forced to be based on several independent samples. Third, we discuss a possibility of relaxing the condition on using independent samples in case of sufficiently large sample sizes.

Dependency between the confidence intervals for α_k^N

Notice that for any given $k \in \mathcal{N}$ the events E^k and E^l are dependent **iff** any two events from the sets $\left\{E^k, E_l^k, E_r^k\right\}$ and $\left\{E^l, E_l^l, E_r^l\right\}$, respectively, are dependent. Example 20 shows that it is vital to have the *c. i.* of α_k^N and α_l^N pairwise independent for any

$k, l \in \mathcal{N}$. Below we show the dependency of these *c. i.* when they are derived from the *same* sample $\overrightarrow{\mathbf{P}_N}$ of independent observations. This is done under the assumptions of $\alpha_k^N \notin \{0, 1\}$ which is not significant because in practice the values of α_k^N are unknown.

For any finite sample size $M \in \mathbb{N}_{\geq 3}$, Proposition 21 below proves that the above-mentioned *c. i.* are *dependent*. Note that in practice the sample size M is taken to be significantly larger than 3.

Proposition 21 *Let* $\overrightarrow{\mathbf{P}_N} = \left(\mathbf{P}_N^1, \ldots, \mathbf{P}_N^M\right)$ *be a sample of* $M \in \mathbb{N}_{\geq 3}$ *independent observations. For any* $k, l \in \mathcal{N}$, $k \neq l$ *and* $\alpha_k^N, \alpha_l^N \notin \{0, 1\}$ *the* c. i. *of* α_k^N *and* α_l^N *are dependent.*

Proof See the proof of Proposition 54 from Appendix C.1.1. □

The *c. i.* are derived using Central Limit Theorem 15 where the limit of $M \to \infty$ is considered. Therefore, if the *c. i.* are independent in the limit, we could say that they are *"approximately independent"*, for sufficiently large values of M. Thus, by means of Proposition 22 below, we also prove the dependency of the *c. i.* in the limit. The latter is done for the case of known variance σ_k^N of the *r. v.* $f_k\left(\mathbf{P}_N\right)$, where $k \in \mathcal{N}$. The proof is based on the multi-dimensional Central Limit Theorem 55 (Appendix C.1.1) and indicates that the dependency between the *c. i.* is also preserved when the sample variances V_k^N are used.

Proposition 22 *Let* $\overrightarrow{\mathbf{P}_N} = \left(\mathbf{P}_N^1, \ldots, \mathbf{P}_N^M\right)$ *be a sample of* $M \in \mathbb{N}_{\geq 2}$ *independent observations, and* $\alpha_k^N \notin \{0, 1\}$ *for any* $k \in \mathcal{N}$. *The* c. i. *of* α_k^N *and* α_l^N *for any* $k, l \in \mathcal{N}$ *and* $k \neq l$, *derived using Equation (6.2) with* $\sigma_k^N = Var\left[f_k\left(\mathbf{P}_N^i\right)\right]$ *used in place of* $\overline{V}_k^N$, *are dependent in the limit of* $M \to \infty$.

Proof See the proof of Proposition 59 from Appendix C.1.1. □

Assume, there are two independent samples $\overrightarrow{\mathbf{P}_N}$ and $\overrightarrow{\mathbf{P}'_N}$. Due to the independence of the samples we can conclude that there is no dependency between the *c. i.* under consideration, if each of the *c. i.* in the pair is based on its own sample. For example, the *c. i.* of α_g^N and α_t^N are independent if the *c. i.* of α_g^N is based on $\overrightarrow{\mathbf{P}_N}$ and the *c. i.* of α_t^N on $\overrightarrow{\mathbf{P}'_N}$. Therefore, we suggest to use several independent samples when deriving the *c. i.* of α_g. The only drawback is that we have to do more simulation runs, but later we will see that for sufficiently large values of M this requirement can be relaxed.

Confidence intervals for α_g

The following theorem gives us the *c. i.* of α_g, imposed by Proposition 19.

Theorem 23 *For independent samples* $\overrightarrow{\mathbf{P}_N}$, $\overrightarrow{\mathbf{P}'_N}$ *and* $\overrightarrow{\mathbf{P}''_N}$ *of* $M \in \mathbb{N}_{\geq 2}$ *independent*

observations each, and the c. i. *of* α_k^N *for all* $k \in \mathcal{N}$ *with confidence* $1-\beta$*, we have:*

$$Prob\left(A_l^g\left(\overrightarrow{\mathbf{P}_N}\right) \leq \alpha_g \leq A_r^g\left(\overrightarrow{\mathbf{P}_N}\right) + A_r^t\left(\overrightarrow{\mathbf{P}'_N}\right)\right) \succeq (1-\beta)\cdot\left(1-\frac{\beta}{2}\right), \quad (6.10)$$

$$Prob\left(1-\left(A_r^b\left(\overrightarrow{\mathbf{P}_N}\right) + A_r^t\left(\overrightarrow{\mathbf{P}'_N}\right)\right) \leq \alpha_g \leq 1-A_l^b\left(\overrightarrow{\mathbf{P}_N}\right)\right) \succeq (1-\beta)\cdot\left(1-\frac{\beta}{2}\right), \quad (6.11)$$

$$Prob\left(A_l^g\left(\overrightarrow{\mathbf{P}_N}\right) \leq \alpha_g \leq A_r^{g,t}\left(\overrightarrow{\mathbf{P}'_N}\right)\right) \succeq \left(1-\frac{\beta}{2}\right)^2, \quad (6.12)$$

$$Prob\left(1-A_r^{b,t}\left(\overrightarrow{\mathbf{P}_N}\right) \leq \alpha_g \leq 1-A_l^b\left(\overrightarrow{\mathbf{P}'_N}\right)\right) \succeq \left(1-\frac{\beta}{2}\right)^2, \quad (6.13)$$

$$Prob\left(A_l^g\left(\overrightarrow{\mathbf{P}_N}\right) \leq \alpha_g \leq 1-A_l^b\left(\overrightarrow{\mathbf{P}'_N}\right)\right) \succeq \left(1-\frac{\beta}{2}\right)^2, \quad (6.14)$$

$$Prob\left(1-A_r^{b,t}\left(\overrightarrow{\mathbf{P}_N}\right) \leq \alpha_g \leq A_r^{g,t}\left(\overrightarrow{\mathbf{P}'_N}\right)\right) \succeq \left(1-\frac{\beta}{2}\right)^2, \quad (6.15)$$

$$Prob\left(1-\left(A_r^b\left(\overrightarrow{\mathbf{P}''_N}\right) + A_r^t\left(\overrightarrow{\mathbf{P}'_N}\right)\right) \leq \alpha_g \leq A_r^g\left(\overrightarrow{\mathbf{P}_N}\right) + A_r^t\left(\overrightarrow{\mathbf{P}'_N}\right)\right) \succeq \left(1-\frac{\beta}{2}\right)^3, \quad (6.16)$$

$$Prob\left(1-\left(A_r^b\left(\overrightarrow{\mathbf{P}''_N}\right) + A_r^t\left(\overrightarrow{\mathbf{P}_N}\right)\right) \leq \alpha_g \leq A_r^{g,t}\left(\overrightarrow{\mathbf{P}'_N}\right)\right) \succeq \left(1-\frac{\beta}{2}\right)^3, \quad (6.17)$$

$$Prob\left(1-A_r^{b,t}\left(\overrightarrow{\mathbf{P}''_N}\right) \leq \alpha_g \leq A_r^g\left(\overrightarrow{\mathbf{P}_N}\right) + A_r^t\left(\overrightarrow{\mathbf{P}'_N}\right)\right) \succeq \left(1-\frac{\beta}{2}\right)^3. \quad (6.18)$$

Proof See the proof of Theorem 60 from Appendix C.1.2. □

The right-hand side of each of these *c. i.* is the confidence, called ξ. The way these *c. i.* are derived is illustrated by Example 20. In the following, with respect to the *c. i.* given by Theorem 23, we will discuss two important questions: *(i)* is it possible to use just one sample in these *c. i.*, i.e. to take $\overrightarrow{\mathbf{P}_N} = \overrightarrow{\mathbf{P}'_N} = \overrightarrow{\mathbf{P}''_N}$; *(ii)* can we choose the best among the provided *c. i.* Below we discuss the former question, the latter one is tackled in the next section.

Using one sample of observation

First, let us make a remarkable observation based on *the strong law of large numbers for Bernoulli Trials* [128].

Proposition 24 *For any* $N \in \mathbb{N}_{\geq 0}$*, two independent samples* $\overrightarrow{\mathbf{P}_N}$ *and* $\overrightarrow{\mathbf{P}'_N}$ *of* $M \in \mathbb{N}_{>0}$ *independent observations each, the following limit holds* a. s.

$$\lim_{M\to\infty}\left(\left|\frac{\Gamma^k\left(\overrightarrow{\mathbf{P}_N}\right)}{M} - \frac{\Gamma^k\left(\overrightarrow{\mathbf{P}'_N}\right)}{M}\right|\right) = 0$$

Proof See the proof of Proposition 61 from Appendix C.1.2. □

This proposition states that for two independent vectors $\overrightarrow{\mathbf{P}_N}$ and $\overrightarrow{\mathbf{P}'_N}$ of M *i. i. d. r. v.* each, the proportion of k-type events in samples $\overrightarrow{\mathbf{P}_N}$ and $\overrightarrow{\mathbf{P}'_N}$ is *a. s.* the same if M goes to ∞. As a consequence, for a fixed confidence $1-\beta$, we have the following theorem.

Theorem 25 *For any $N \in \mathbb{N}_{\geq 0}$, confidence $1-\beta$, two independent samples $\overrightarrow{\mathbf{P}_N}$ and $\overrightarrow{\mathbf{P}'_N}$ of $M \in \mathbb{N}_{>0}$ independent observations each, the following holds:*

$$Prob\left(\lim_{M\to\infty}\left(\left|A_l^k\left(\overrightarrow{\mathbf{P}_N}\right) - A_l^k\left(\overrightarrow{\mathbf{P}'_N}\right)\right|\right) = 0\right) = 1,$$
$$Prob\left(\lim_{M\to\infty}\left(\left|A_r^k\left(\overrightarrow{\mathbf{P}_N}\right) - A_r^k\left(\overrightarrow{\mathbf{P}'_N}\right)\right|\right) = 0\right) = 1.$$

Proof See the proof of Theorem 64 from Appendix C.1.2. □

This theorem implies that, for sufficiently large M, Equations (6.10) to (6.18) are likely to provide the same *c. i.* bounds, both using different samples or just one sample (i.e. taking $\overrightarrow{\mathbf{P}_N} = \overrightarrow{\mathbf{P}'_N} = \overrightarrow{\mathbf{P}''_N}$).

In practice we are not going to use the results of Theorem 25 because of the following reasons: *(i)* we do not know how large the value of M should be in order to start using one sample; *(ii)* in case we decide to use one sample, the confidence ξ should be re-derived. For example, in Equation (6.12) we have $\xi = \left(1-\frac{\beta}{2}\right)^2$, but this is under the assumption of using two independent samples $\overrightarrow{\mathbf{P}_N}$ and $\overrightarrow{\mathbf{P}'_N}$.

6.2.4 Choosing the best *c. i.* for $Prob\,(s_0,\ \mathcal{A}\ \mathrm{U}\ \mathcal{G})$

At this point we would like to reduce the number of *c. i.* for α_g. First notice that, due to Lemma 26 below, Equations (6.12) through (6.15) are equivalent. Thus we can leave just one of them, for example Equation (6.12).

Lemma 26 *For a fixed confidence $1-\beta$ and a sample $\overrightarrow{\mathbf{P}_N}$ of $M \in \mathbb{N}_{\geq 2}$ observations:*

$$A_l^g\left(\overrightarrow{\mathbf{P}_N}\right) = 1 - A_r^{b,t}\left(\overrightarrow{\mathbf{P}_N}\right), \quad A_r^{g,t}\left(\overrightarrow{\mathbf{P}_N}\right) = 1 - A_l^b\left(\overrightarrow{\mathbf{P}_N}\right). \tag{6.19}$$

Proof See the proof of Lemma 65 from Appendix C.1.2. □

Further, let us first analyze the remaining equations, assuming that $\overrightarrow{\mathbf{P}_N} = \overrightarrow{\mathbf{P}'_N} = \overrightarrow{\mathbf{P}''_N}$.

Lemma 27 *For a fixed confidence $1-\beta$, $M \in \mathbb{N}_{\geq 2}$, $N \in \mathbb{N}_{\geq 0}$ and sample $\overrightarrow{\mathbf{P}_N}$ of M observations the following holds:*

$$A_r^{g,t}\left(\overrightarrow{\mathbf{P}_N}\right) \leq A_r^g\left(\overrightarrow{\mathbf{P}_N}\right) + A_r^t\left(\overrightarrow{\mathbf{P}_N}\right), \quad A_r^{b,t}\left(\overrightarrow{\mathbf{P}_N}\right) \leq A_r^b\left(\overrightarrow{\mathbf{P}_N}\right) + A_r^t\left(\overrightarrow{\mathbf{P}_N}\right).$$

Proof See the proof of Lemma 66 from Appendix C.1.2. □

Lemma 27 in combination with Equation (6.19) implies that Equation (6.12) provides the *c. i.* that is enclosed in the *c. i.* provided by other equations. Therefore, considering Algorithm 3, it is the best equation for us when using one sample of observations.

Now we can move on to the case of different samples, i. e. $\overrightarrow{\mathbf{P}_N} \neq \overrightarrow{\mathbf{P}'_N} \neq \overrightarrow{\mathbf{P}''_N}$. Let $A_l\left(\overrightarrow{\mathbf{P}_N}, \overrightarrow{\mathbf{P}'_N}, \overrightarrow{\mathbf{P}''_N}\right) \leq \alpha_g \leq A_r\left(\overrightarrow{\mathbf{P}_N}, \overrightarrow{\mathbf{P}'_N}, \overrightarrow{\mathbf{P}''_N}\right)$ be a general form of the *c. i.* for α_g, then Table 6.1 gives the analysis of its borders with respect to the borders of the *c. i.* for α_k^N. We provide results for Equations (6.10) to (6.12) only [6]. It is easy to see that the tighter the *c. i.* of α_k^N for all $k \in \mathcal{N}$, the tighter the induced *c. i.* of α_g.

[6] For the remaining equations the results are the same.

Equation:	(6.10)		(6.11)		(6.12)	
$A_l\left(\overrightarrow{\mathbf{P}_N}, \overrightarrow{\mathbf{P}'_N}, \overrightarrow{\mathbf{P}''_N}\right)$	$A_l^g\left(\overrightarrow{\mathbf{P}_N}\right)$	$\nearrow^a$	$A_r^b\left(\overrightarrow{\mathbf{P}_N}\right)$	$\searrow^b$	$A_l^g\left(\overrightarrow{\mathbf{P}_N}\right)$	$\nearrow$
			$A_r^t\left(\overrightarrow{\mathbf{P}'_N}\right)$	$\searrow$		
	$\nearrow$		$\nearrow$		$\nearrow$	
$A_r\left(\overrightarrow{\mathbf{P}_N}, \overrightarrow{\mathbf{P}'_N}, \overrightarrow{\mathbf{P}''_N}\right)$	$A_r^g\left(\overrightarrow{\mathbf{P}_N}\right)$	$\searrow$	$A_l^b\left(\overrightarrow{\mathbf{P}_N}\right)$	$\nearrow$	$A_r^{g,t}\left(\overrightarrow{\mathbf{P}'_N}\right)$	$\searrow$
	$A_r^t\left(\overrightarrow{\mathbf{P}'_N}\right)$	$\searrow$				
	$\searrow$		$\searrow$		$\searrow$	

[a] Increases.
[b] Decreases.

Table 6.1: The dependency between the *c. i.* of α_g and α_k^N

For instance, the *c. i.* borders given by Equation 6.10 are $A_l\left(\overrightarrow{\mathbf{P}_N}, \overrightarrow{\mathbf{P}'_N}, \overrightarrow{\mathbf{P}''_N}\right) = A_l^g\left(\overrightarrow{\mathbf{P}_N}\right)$ and $A_r\left(\overrightarrow{\mathbf{P}_N}, \overrightarrow{\mathbf{P}'_N}, \overrightarrow{\mathbf{P}''_N}\right) = A_r^g\left(\overrightarrow{\mathbf{P}_N}\right) + A_r^t\left(\overrightarrow{\mathbf{P}'_N}\right)$, Table 6.1 indicates that with the increase ($\nearrow$) of the left border $A_l^g\left(\overrightarrow{\mathbf{P}_N}\right)$ we have an increase of $A_l\left(\overrightarrow{\mathbf{P}_N}, \overrightarrow{\mathbf{P}'_N}, \overrightarrow{\mathbf{P}''_N}\right)$ and with the decrease ($\searrow$) of the right borders $A_r^g\left(\overrightarrow{\mathbf{P}_N}\right)$ and $A_r^t\left(\overrightarrow{\mathbf{P}'_N}\right)$ we have a decrease of $A_r\left(\overrightarrow{\mathbf{P}_N}, \overrightarrow{\mathbf{P}'_N}, \overrightarrow{\mathbf{P}''_N}\right)$. This means that the tighter the *c. i.* of α_g^N and α_t^N the tighter is the *c. i.* of α_g.

The potential threat of tightening the *c. i.* is that since the left and right borders are based on different samples we may get $A_l\left(\overrightarrow{\mathbf{P}_N}, \overrightarrow{\mathbf{P}'_N}, \overrightarrow{\mathbf{P}''_N}\right) > A_r\left(\overrightarrow{\mathbf{P}_N}, \overrightarrow{\mathbf{P}'_N}, \overrightarrow{\mathbf{P}''_N}\right)$ resulting in an invalid interval. Note that the problem *can not* be cured by simply swapping the samples, e. g., if in Equation (6.12) we get $A_l^g\left(\overrightarrow{\mathbf{P}_N}\right) > A_r^{g,t}\left(\overrightarrow{\mathbf{P}'_N}\right)$, then by swapping $\overrightarrow{\mathbf{P}_N}$ and $\overrightarrow{\mathbf{P}'_N}$ we get $A_l^g\left(\overrightarrow{\mathbf{P}'_N}\right) \leq A_r^{g,t}\left(\overrightarrow{\mathbf{P}_N}\right)$. The reason for that, along with the possible solutions, will be given later.

Equations	**# samples**	$\mathbf{1-\beta}$	**Range of** ξ
(6.10), (6.11)	2	$\frac{1}{2}\cdot\left(\sqrt{8\cdot\xi+1}-1\right)$	$\mathbb{R}_{[0,1]}$
(6.12) to (6.15)	2	$2\cdot\sqrt{\xi}-1$	$\mathbb{R}_{[0.25,1]}$
(6.16) to (6.18)	3	$2\cdot\sqrt[3]{\xi}-1$	$\mathbb{R}_{[0.125,1]}$

Table 6.2: The required parameters for deriving the *c. i.* of α_g with the confidence ξ

Before we proceed with the further analysis, it is important to note that Lemma 27 has its analog for the case of distinct samples.

Lemma 28 *For a fixed confidence* $1-\beta$, $N \in \mathbb{N}_{\geq 0}$, *and a finite-state DTMC* $\mathbf{P}$, *with a positive probability there exist independent samples* $\overrightarrow{\mathbf{P}_N}$, $\overrightarrow{\mathbf{P}'_N}$ *and* $\overrightarrow{\mathbf{P}''_N}$ *of* $M \in \mathbb{N}_{\geq 2}$ *independent observations each, such that:*

$$A_r^{g,t}\left(\overrightarrow{\mathbf{P}_N}\right) \leq A_r^{g}\left(\overrightarrow{\mathbf{P}''_N}\right) + A_r^{t}\left(\overrightarrow{\mathbf{P}'_N}\right), \quad \textit{and} \quad A_r^{b,t}\left(\overrightarrow{\mathbf{P}_N}\right) \leq A_r^{b}\left(\overrightarrow{\mathbf{P}''_N}\right) + A_r^{t}\left(\overrightarrow{\mathbf{P}'_N}\right). \tag{6.20}$$

Proof See the proof of Lemma 67 from Appendix C.1.2. □

The probability of Equation 6.20 holding is expected to increase with the growth of M due to Theorem 25, that assures the convergence of the *c. i.* borders based on independent samples.

Notice that for fixed M and N the width of the *c. i.* for α_g depends on the confidence ξ and used samples. Let us study this dependency by discussing the following cases:

1. **Fixed ξ:** Lemma 28 in combination with Lemma 26, similarly to the case of taking $\overrightarrow{\mathbf{P}_N} = \overrightarrow{\mathbf{P}'_N} = \overrightarrow{\mathbf{P}''_N}$, suggests that with a positive probability Equation (6.12) provides the tightest *c. i.* for α_g.
2. **Fixed samples:** For any fixed ξ, the smaller value of $1-\beta$ provides the tighter *c. i.* of α_k^N which in its turn makes the *c. i.* of α_g tighter. The dependencies of $1-\beta$ from ξ for all given equations are given in Table 6.2 and depicted in Figure 6.2. It is clear that for any fixed ξ the values of $1-\beta$ are the smallest for Equation (6.12) thus making it the most attractive to use.

This leaves Equation (6.12) to be the most preferable to use, considering that:

1. The dependency of $1-\beta$ from ξ is of utmost importance, because from Section 5.3 we know that the *c. i.* width increases rapidly when the confidence reaches one.
2. The use of Equations (6.16) to (6.18) seems to be impractical, since they are based on three independent samples.
3. The overall user-defined confidence ξ, in case of Equation (6.12), belongs to $\mathbb{R}_{[0.25,1.0]}$ but not $\mathbb{R}_{[0,1]}$. This limitation is not significant because in practice (typically) we would like to have a *c. i.* with a confidence higher than 0.5.

Now, after having obtained only one *c. i.*, namely the one defined by Equation (6.12):

$$\textit{Prob}\left(A_l^g\left(\overrightarrow{\mathbf{P}_N}\right) \leq \alpha_g \leq A_r^{g,t}\left(\overrightarrow{\mathbf{P}'_N}\right)\right) \succeq \left(1-\frac{\beta}{2}\right)^2,$$

let us consider an example.

Example 21 *For the DTMC of Example 19 let us compute the* 95% c. i. *(i. e.* $\xi = 0.95$*) for* α_g *using Equation (6.12). According to Table 6.2 we should choose:*

$$1-\beta = 2\cdot\sqrt{\xi} - 1 = 0.949358869,$$

meaning that $\beta = 0.050641131$, *and* $\widetilde{z_n}\,(0.050641131) \approx 1.96$.

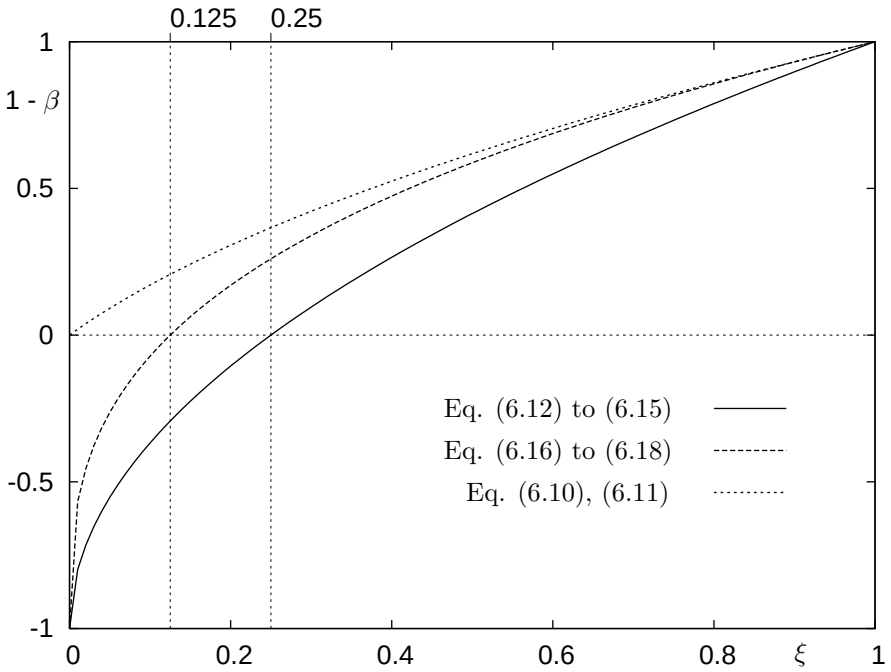


Figure 6.2: The confidence $1-\beta$ versus the confidence ξ

	Γ_N^g	$A_l^g(.)$	$\Gamma_N^{g,t}$	$A_r^{g,t}(.)$
$\overrightarrow{\mathbf{P}_3}$	6	0.27993334	7	0.999394945
$\overrightarrow{\mathbf{P}'_3}$	5	0.173333333	6	0.92006666

Table 6.3: Computation of $A_l(.)$ and $A_r(.)$ for the *c. i.* of α_g

Assume, for $M = 10$ *and* $N = 3$ *we obtained the following two samples:*

$$\overrightarrow{\mathbf{P}_3} = (0,2,1,1,2,1,1,1,2,1), \quad \overrightarrow{\mathbf{P}'_3} = (2,2,1,1,1,0,1,1,2,2).$$

As before $\mathcal{G} = \{1\}$ *and* $\mathcal{A} = \{0\}$. *Table 6.3 contains values of* Γ_N^k, *for* $k \in \{\{g\},\{g,t\}\}$, $A_l^g(.)$ *and* $A_r^{g,t}(.)$ *on both samples.*

Now it is easy to see that Equation (6.12) contains two c. i. *instead of one. This is because the sample vectors can be swapped when deriving* $A_l^g(.)$ *and* $A_r^{g,t}(.)$*:*

$$Prob\left(A_l^g\left(\overrightarrow{\mathbf{P}_3}\right) \le \alpha_g \le A_r^{g,t}\left(\overrightarrow{\mathbf{P}'_3}\right)\right) = 0.95,$$
$$Prob\left(A_l^g\left(\overrightarrow{\mathbf{P}'_3}\right) \le \alpha_g \le A_r^{g,t}\left(\overrightarrow{\mathbf{P}_3}\right)\right) = 0.95.$$

Swapping the samples

As it was noted in Example 21, Equation (6.12) actually provides us with two *c. i.*, namely:

$$Prob\left(A_l^g\left(\overrightarrow{\mathbf{P}_N}\right) \leq \alpha_g \leq A_r^{g,t}\left(\overrightarrow{\mathbf{P}'_N}\right)\right) \succeq \left(1-\frac{\beta}{2}\right)^2, \text{ and} \tag{6.21}$$

$$Prob\left(A_l^g\left(\overrightarrow{\mathbf{P}'_N}\right) \leq \alpha_g \leq A_r^{g,t}\left(\overrightarrow{\mathbf{P}_N}\right)\right) \succeq \left(1-\frac{\beta}{2}\right)^2. \tag{6.22}$$

Notice, that these *c. i.* are defined by two *c. i.*, based on one sample each:

$$I_N = \left[A_l^g\left(\overrightarrow{\mathbf{P}_N}\right), A_r^{g,t}\left(\overrightarrow{\mathbf{P}_N}\right)\right], \text{ and } I'_N = \left[A_l^g\left(\overrightarrow{\mathbf{P}'_N}\right), A_r^{g,t}\left(\overrightarrow{\mathbf{P}'_N}\right)\right]. \tag{6.23}$$

Below, we first show that Algorithm 3 does not give "contradictory" answers on the *c. i.* defined by Equations (6.21) and (6.22). Then we discuss how to resolve the choice between these *c. i.*

Definition 23 *We say that Algorithm 3 gives a* non-contradictory answer *for two* c. i. $\left[A_l^1, A_r^1\right]$ *and* $\left[A_l^2, A_r^2\right]$ *if for any* $p \in \mathbb{N}_{[0,1]}$ *and* $\bowtie \in \{<, \leq, >, \geq\}$ *we never have answers TRUE for one* c. i. *and FALSE for another.*

Lemma 29 *For two* c. i. $\left[A_l^1, A_r^1\right]$ *and* $\left[A_l^2, A_r^2\right]$ *such that* $A_l^2 \leq A_r^1$ *and* $A_l^1 \leq A_r^2$ *Algorithm 3 gives a non-contradictory answer.*

Proof See the proof of Lemma 68 from Appendix C.1.2. □

From Lemma 29 it follows that Algorithm 3 does not provide contradictory answers for the *c. i.* given by Equations (6.21) and (6.22). The latter is because $A_l^g\left(\overrightarrow{\mathbf{P}_N}\right) \leq A_r^{g,t}\left(\overrightarrow{\mathbf{P}_N}\right)$ and $A_l^g\left(\overrightarrow{\mathbf{P}'_N}\right) \leq A_r^{g,t}\left(\overrightarrow{\mathbf{P}'_N}\right)$, i. e. the initial conditions of the lemma are satisfied.

From the statistical point of view, the choice between Equations (6.21) and (6.22) can be resolved only once, prior to the *c. i.* computation. Otherwise, if we decide what *c. i.* to use based on the knowledge of one of them being "better" (tighter) we are affecting the confidence. Therefore, taking into the account that the provided *c. i.* are equivalent up to the permutation of samples, without loss of generality, we can always use the *c. i.* given by Equation (6.21).

Now, when we have the *c. i.* for α_g in place, recall that there are still some uncertainties in our approach. Namely, when computing the sequential[7] *c. i.* of α_g we would like to know if it is more advantageous to increase the sample size M or the simulation length N.

6.2.5 The *c. i.* dependency on the sample size and the simulation depth

Below we consider the *c. i.* of α_g given by Equation (6.21). The main goal of this section is to investigate the dependency of the *c. i.* width from the parameters, such

[7] As it is required for Algorithm 3, see the reasoning in Section 6.1.

as sample size M and simulation depth N. The latter can allow us to optimize the performance of the model-checking procedure described in the next section.

Let us start our analysis with considering the left $A_l^g(.)$ and right $A_r^{g,t}(.)$ border of the *c. i.* for α_g. These borders stem either from Equation (6.4) or (6.6). Also, as it was noted in Section 6.2.2 they can be seen as functions of Γ_N^k:

$$A_l^k\left(\overrightarrow{\mathbf{P}_N}\right) = A_l^k\left(\Gamma_N^k\right), \qquad A_r^k\left(\overrightarrow{\mathbf{P}_N}\right) = A_r^k\left(\Gamma_N^k\right),$$
$$\widetilde{A}_l^k\left(\overrightarrow{\mathbf{P}_N}\right) = \widetilde{A}_l^k\left(\Gamma_N^k\right), \qquad \widetilde{A}_r^k\left(\overrightarrow{\mathbf{P}_N}\right) = \widetilde{A}_r^k\left(\Gamma_N^k\right).$$

In this case the borders have N as an explicit parameter which only influences the value of $\Gamma_N^k = \Gamma^k\left(\overrightarrow{\mathbf{P}_N}\right)$. This always us to first study the behavior of the *c. i.* borders with respect to Γ_N^k, and then explore the dependency of Γ_N^k on $\overrightarrow{\mathbf{P}_N}$.

We already know that the *c. i.* of α_g is defined by the two intervals I_N and $I_N^{'}$. Thus, the former one can be analyzed by considering the latter two. Note that in case of taking $\overrightarrow{\mathbf{P}_N} = \overrightarrow{\mathbf{P}'_N}$ we have $I_N = I_N^{'}$, uniquely defining behavior of the *c. i.* given by Equation (6.12).

In the following, we first analyze the behavior of the *c. i.* with respect to the sample size and then the simulation depth.

The dependency on sample size M. In order to understand the behavior of borders $A_l^k\left(\Gamma_N^k\right)$ and $A_r^k\left(\Gamma_N^k\right)$ with respect to M consider the following lemma.

Lemma 30 *For a fixed confidence $1-\beta$, $k \in \mathcal{N}$ and $M \in \mathbb{N}_{\geq 2}$ the following holds:*

$$\overline{X}_k^N - \sqrt{2}\cdot\frac{\widetilde{z_n}(\beta)}{\sqrt{M}} \leq A_l^k\left(\Gamma_N^k\right) \leq \overline{X}_k^N \leq A_r^k\left(\Gamma_N^k\right) \leq \overline{X}_k^N + \sqrt{2}\cdot\frac{\widetilde{z_n}(\beta)}{\sqrt{M}}.$$

Proof See the proof of Lemma 63 from Appendix C.1.2. □

Note that the behavior of $\widetilde{A}_l^k\left(\Gamma_N^k\right)$ and $\widetilde{A}_r^k\left(\Gamma_N^k\right)$ is similar, namely:

$$\widetilde{X}_k^N - \sqrt{2}\cdot\frac{\widetilde{z_n}(\beta)}{\sqrt{M}} \leq \widetilde{A}_l^k\left(\Gamma_N^k\right) \leq \widetilde{X}_k^N \leq \widetilde{A}_r^k\left(\Gamma_N^k\right) \leq \widetilde{X}_k^N + \sqrt{2}\cdot\frac{\widetilde{z_n}(\beta)}{\sqrt{M}}$$

It is clear now, that the widths of the intervals I_N and $I_N^{'}$ decrease proportionally to $\frac{1}{\sqrt{M}}$ with the increase of M. On the other hand, the center points of these intervals may shift with the increase of M, making I_N and $I_N^{'}$ move against each. These means that the *c. i.* of α_g, can shift and become both shorter or wider with the increase of M.

Example 22 *Consider again Example 21 and increase the sample size M by 10 via extending the samples $\overrightarrow{\mathbf{P}_3}$ and $\overrightarrow{\mathbf{P}_3'}$ by 10 observations each:*

$$\overrightarrow{\mathbf{P}_3} = (0,2,1,1,2,1,1,1,2,1,\mathit{0},\mathit{0},\mathit{0},\mathit{0},\mathit{0},\mathit{2},\mathit{2},\mathit{2},\mathit{2},\mathit{2}),$$
$$\overrightarrow{\mathbf{P}_3'} = (2,2,1,1,1,0,1,1,2,2,\mathit{1},\mathit{0},\mathit{0},\mathit{0},\mathit{0},\mathit{0},\mathit{2},\mathit{2},\mathit{2},\mathit{2}).$$

These samples are obtained via improper simulation because the observations do not comply to the underlying probability distribution. Nevertheless such samples can occur

	Γ_N^g	$A_l^g(.)$	$\Gamma_N^{g,t}$	$A_r^{g,t}(.)$
$\overrightarrow{\mathbf{P}_3}$	6	**0.093942267**	12	0.820284983
$\overrightarrow{\mathbf{P}_3'}$	7	0.135528311	12	**0.820284983**

Table 6.4: Computation of $A_l(.)$ and $A_r(.)$ for the *c. i.* of α_g

and thus we take them to illustrate a possible increase in the width of the c. i. *of* α_g *due to the increase of* M*.*

Table 6.4 contains some values of Γ_N^k *on both samples. Equation (6.21) gives us the needed* c. i.*, that is* [0.093942267, 0.820284983]*. Notice, that the width of the* c. i. *chosen in Example 21 is* 0.64013332 *whereas now it is larger, namely* 0.72634272*.*

Although the increase of the *c. i.* width described in the example above is possible, let us note that increasing M should eventually tighten the *c. i.* of α_g due to the convergence of the *c. i.* borders (cf. Section 6.2.3). Also, recall that for the correct *c. i.* of α_g^N and $\alpha_{g,t}^N$ we have:

$$A_l^g\left(\overrightarrow{\mathbf{P}_N}\right) \leq \alpha_g^N \leq \alpha_g \leq \alpha_{g,t}^N \leq A_r^{g,t}\left(\overrightarrow{\mathbf{P}_N}\right).$$

Thus, for any M the width of the correct *c. i.* of α_g can not be smaller than $\alpha_{g,t}^N - \alpha_g^N$.

The dependency on simulation depth N**.** Below we first analyze the borders $A_l^k\left(\overrightarrow{\mathbf{P}_N}\right)$ and $A_r^k\left(\overrightarrow{\mathbf{P}_N}\right)$ given by Equation (6.4), and then $\widetilde{A}_l^k\left(\overrightarrow{\mathbf{P}_N}\right)$ and $\widetilde{A}_r^k\left(\overrightarrow{\mathbf{P}_N}\right)$ as provided by Equation (6.6).

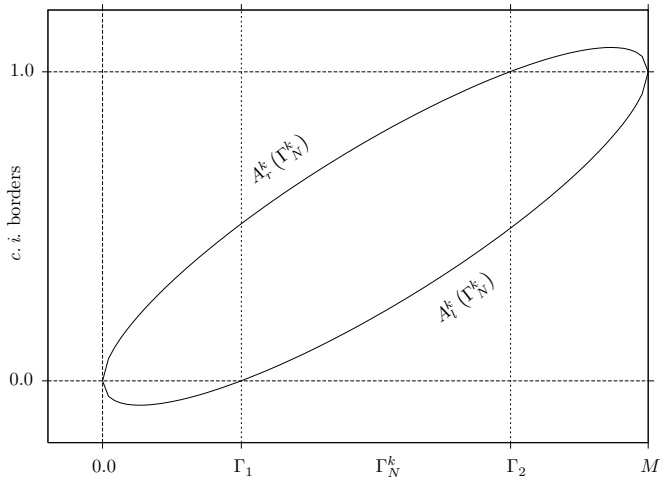


Figure 6.3: The behavior of $A_l^k\left(\Gamma_N^k\right)$ and $A_r^k\left(\Gamma_N^k\right)$

Figure 6.3 shows the dependency of $A_r^k\left(\Gamma_N^k\right)$ and $A_l^k\left(\Gamma_N^k\right)$ on Γ_N^k. Their values increase with the increase of $\Gamma_N^k \in \mathbb{N}_{(0,\Gamma_2)}$ and $\Gamma_N^k \in \mathbb{N}_{(\Gamma_1,M)}$ correspondingly. Here

Γ_1 and Γ_2 are such that $A_l^k(\Gamma_1) = 0$, $A_r^k(\Gamma_2) = 1$. These results are reflected by Lemma 31 and Proposition 32.

Lemma 31 *For* $A_l^k\left(\overrightarrow{\mathbf{P}_N}\right)$ *and* $A_r^k\left(\overrightarrow{\mathbf{P}_N}\right)$ *given by Equation (6.4) let* $k \in \mathcal{N}$, $\widetilde{z_n}(\beta) \geq 0$, $M \in \mathbb{N}_{\geq 2}$ *and* $\Gamma_N^k \in \mathbb{N}_{[0,M]}$ *then the following holds:*

- $A_l^k\left(\Gamma_N^k\right) = 0$ **iff** $\Gamma_N^k = \frac{(\widetilde{z_n}(\beta))^2 \cdot M}{(\widetilde{z_n}(\beta))^2 + M - 1}$ *or* $\Gamma_N^k = 0$.
- $A_r^k\left(\Gamma_N^k\right) = 1$ **iff** $\Gamma_N^k = \frac{M \cdot (M-1)}{(\widetilde{z_n}(\beta))^2 + M - 1}$ *or* $\Gamma_N^k = M$.

Proof See the proof of Proposition 69 from Appendix C.1.3. □

Proposition 32 *For* $A_l^k\left(\overrightarrow{\mathbf{P}_N}\right)$ *and* $A_r^k\left(\overrightarrow{\mathbf{P}_N}\right)$ *given by Equation (6.4) let* $k \in \mathcal{N}$, $\widetilde{z_n}(\beta) \geq 0$, $M \in \mathbb{N}_{\geq 2}$, $\Gamma_N^k \in \mathbb{N}_{(0,M)}$ *then:*

$$\Gamma_1 = \frac{(\widetilde{z_n}(\beta))^2 \cdot M}{(\widetilde{z_n}(\beta))^2 + M - 1}, \quad \textit{and} \quad \Gamma_2 = \frac{M \cdot (M-1)}{(\widetilde{z_n}(\beta))^2 + M - 1}.$$

- $A_l^k\left(\Gamma_N^k\right)$ *is increasing from* 0 *to* 1.0 *with the increase of* $\Gamma_N^k \in \mathbb{N}_{(\Gamma_1, M)}$,
- $A_r^k\left(\Gamma_N^k\right)$ *is increasing from* 0 *to* 1.0 *with the increase of* $\Gamma_N^k \in \mathbb{N}_{(0,\Gamma_2)}$.

Proof See the proof of Proposition 71 from Appendix C.1.3. □

It is easy to see that for a particular sample $\overrightarrow{\mathbf{P}_N}$, Γ_N^g is non-decreasing and $\Gamma_N^{g,t}$ is non-increasing with the increase of N due to the absorbing structure of the considered Markov chain. The increase of N means that for the sample $\overrightarrow{\mathbf{P}_N}$ we simulate the states defined by its observations for $K \in \mathbb{N}_{>0}$ time units and obtain the sample $\overrightarrow{\mathbf{P}_{N+K}}$ that results in Γ_{N+K}^g and $\Gamma_{N+K}^{g,t}$. It is also important to note that for any $N \in \mathbb{N}_{\geq 0}$ we have $\Gamma_N^g \leq \Gamma_N^{g,t}$ and therefore $A_l^g(\Gamma_N^g) \leq A_r^{g,t}\left(\Gamma_N^{g,t}\right)$.

Now we can investigate the behavior of the interval I_N $(I_N^{'})$, defined by Equation (6.23) (cf. page 124). For that consider Figure 6.4 that is a consequence of the facts mentioned above and Figure 6.3. Let $A = A_r^{g,t}\left(\Gamma_N^{g,t}\right)$, $B = A_r^{g,t}(\Gamma_N^g)$, $C = A_l^g\left(\Gamma_N^{g,t}\right)$ and $D = A_l^g(\Gamma_N^g)$ then for any $K \in \mathbb{N}_{\geq 0}$ we have:

$$B \leq A_l^g\left(\Gamma_{N+K}^g\right) \leq A \qquad \text{and} \qquad D \leq A_r^{g,t}\left(\Gamma_{N+K}^{g,t}\right) \leq C,$$

which means that for any $K \in \mathbb{N}_{\geq 0}$ we have:

$$I_{N+K} \subseteq I_N \text{ and similarly } I_{N+K}^{'} \subseteq I_N^{'}. \tag{6.24}$$

Figure 6.5 shows the typical behaviour of $\widetilde{A}_r^k\left(\Gamma_N^k\right)$ and $\widetilde{A}_l^k\left(\Gamma_N^k\right)$ with respect to Γ_N^k. It is very similar to the one exhibited by the borders $A_l^k\left(\Gamma_N^k\right)$ and $A_r^k\left(\Gamma_N^k\right)$ (cf. Figure 6.3). Therefore, we can conclude that the interval inclusion given by Equation (6.24) also holds in case of I_N and $I_N^{'}$ based on $\widetilde{A}_r^k(.)$ and $\widetilde{A}_l^k(.)$.

We are now interested in how the *c. i.* of α_g at epoch $N+K$, with $K \in \mathbb{N}_{\geq 0}$, depends on the *c. i.* of α_g at epoch N. For that we consider the intervals I_N and $I_N^{'}$ as depicted in Figure 6.6.

- **Figure 6.6(a):** the *c. i.* at epoch N is invalid and it remains to be invalid at epoch $N+K$ because $A_l^g\left(\overrightarrow{\mathbf{P}_N}\right) \leq A_l^g\left(\overrightarrow{\mathbf{P}_{N+K}}\right)$ and $A_r^{g,t}\left(\overrightarrow{\mathbf{P}'_{N+K}}\right) \leq A_r^{g,t}\left(\overrightarrow{\mathbf{P}'_N}\right)$.
- **Figure 6.6(b):** the *c. i.* at epoch $N+K$ either becomes invalid or we obtain a tighter *c. i.*

$$\left[A_l^g\left(\overrightarrow{\mathbf{P}_{N+K}}\right),\ A_r^{g,t}\left(\overrightarrow{\mathbf{P}'_{N+K}}\right)\right] \subseteq \left[A_l^g\left(\overrightarrow{\mathbf{P}_N}\right),\ A_r^{g,t}\left(\overrightarrow{\mathbf{P}'_N}\right)\right].$$

- **Figure 6.6(c):** is similar to the case of Figure 6.6(b).

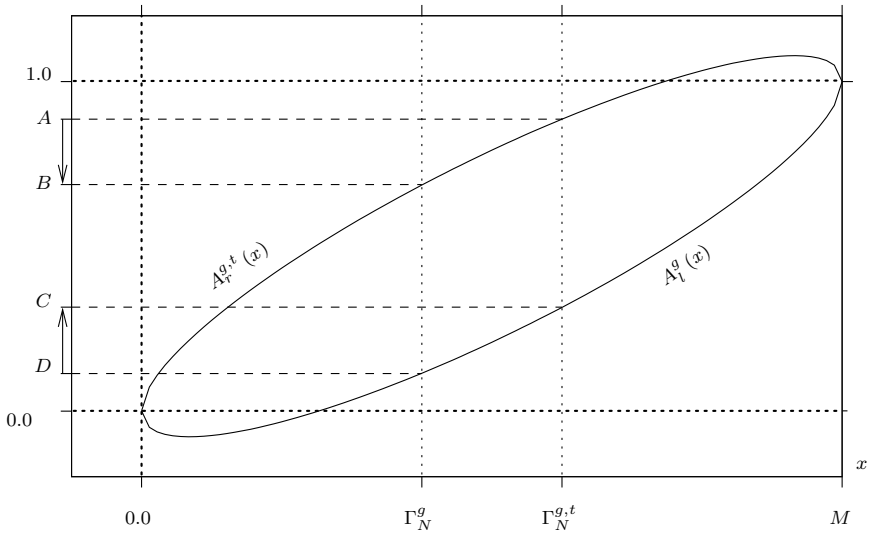


Figure 6.4: The behavior of $A_l^g(x)$ and $A_r^{g,t}(x)$ with $x \in \mathbb{N}_{[0,M]}$

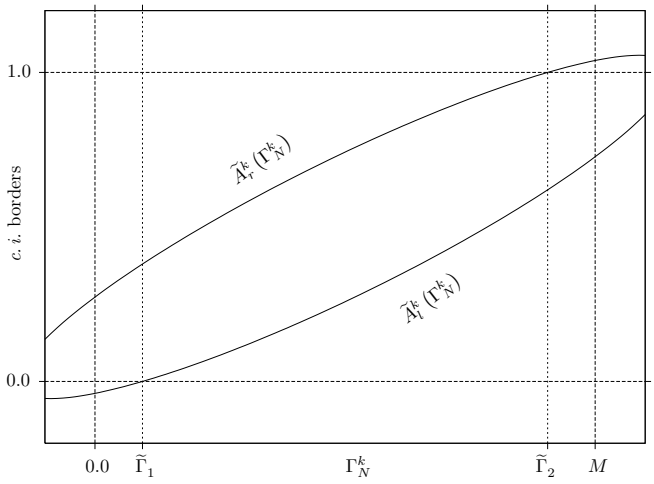
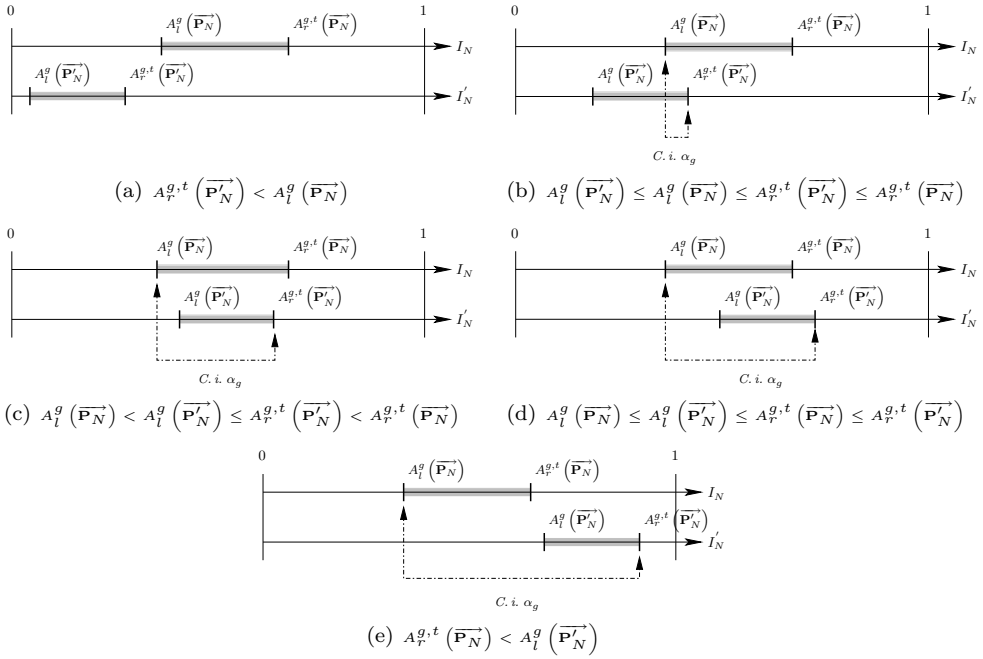


Figure 6.5: The behavior of $\widetilde{A}_l^k\left(\Gamma_N^k\right)$ and $\widetilde{A}_r^k\left(\Gamma_N^k\right)$

- **Figure 6.6(d):** is similar to the case of Figure 6.6(b).
- **Figure 6.6(e):** then, since $A_r^{g,t}\left(\overrightarrow{\mathbf{P}_N}\right) < A_l^g\left(\overrightarrow{\mathbf{P}'_N}\right)$, we always have:

$$\left[A_l^g\left(\overrightarrow{\mathbf{P}_{N+K}}\right),\ A_r^{g,t}\left(\overrightarrow{\mathbf{P}'_{N+K}}\right)\right] \subseteq \left[A_l^g\left(\overrightarrow{\mathbf{P}_N}\right),\ A_r^{g,t}\left(\overrightarrow{\mathbf{P}'_N}\right)\right].$$

It is clearly now that the *c. i.* of α_g is either enclosed in the *c. i.* thereof derived for the previous time point or is invalid (cf. Figures 6.6(a)). Moreover, both Figures 6.6(a) and 6.6(e) correspond to the cases when we do not have a proper *c. i.* of α_g. The latter follows from the fact that correct intervals I_N and I'_N must contain α_g. Therefore, one should use equality $I_N \cap I'_N = \emptyset$ as an indicator of a poor simulation run. Note that such simulation can be cured by increasing the number of observations M, as guaranteed by the convergence of the *c. i.* borders (cf. Section 6.2.3).



(a) $A_r^{g,t}\left(\overrightarrow{\mathbf{P}'_N}\right) < A_l^g\left(\overrightarrow{\mathbf{P}_N}\right)$

(b) $A_l^g\left(\overrightarrow{\mathbf{P}'_N}\right) \leq A_l^g\left(\overrightarrow{\mathbf{P}_N}\right) \leq A_r^{g,t}\left(\overrightarrow{\mathbf{P}'_N}\right) \leq A_r^{g,t}\left(\overrightarrow{\mathbf{P}_N}\right)$

(c) $A_l^g\left(\overrightarrow{\mathbf{P}_N}\right) < A_l^g\left(\overrightarrow{\mathbf{P}'_N}\right) \leq A_r^{g,t}\left(\overrightarrow{\mathbf{P}'_N}\right) < A_r^{g,t}\left(\overrightarrow{\mathbf{P}_N}\right)$

(d) $A_l^g\left(\overrightarrow{\mathbf{P}_N}\right) \leq A_l^g\left(\overrightarrow{\mathbf{P}'_N}\right) \leq A_r^{g,t}\left(\overrightarrow{\mathbf{P}_N}\right) \leq A_r^{g,t}\left(\overrightarrow{\mathbf{P}'_N}\right)$

(e) $A_r^{g,t}\left(\overrightarrow{\mathbf{P}_N}\right) < A_l^g\left(\overrightarrow{\mathbf{P}'_N}\right)$

Figure 6.6: Deriving the *c. i.* of α_g using Equation (6.12)

To conclude the discussion we shall repeat that the increase of M can widen, shrink or shift the *c. i.* of α_g and pushing M to infinity results in a *c. i.* that is not shorter than $\alpha_{g,t}^N - \alpha_g^N$. The increase of N results either in an improper interval (when $I_N \cap I'_N = \emptyset$) or in a subinterval of the original one. Yet, we can not indicate that the increase of N is more advantageous (as opposed to the increase of M), because the *c. i.* borders are functions of Γ_N^k, which change rate solely depends on the Markov chain structure. Thus, for simplicity we suggest to alternate between increasing N and M.

6.2.6 The model-checking procedure

To complete the model-checking procedure for formula $\mathrm{P}_{\bowtie b}(\mathcal{A}\ \mathrm{U}\ \mathcal{G})$ in any initial state $s_0 \in S$ recall that the state space S is divided into three disjoint parts: the "allowed" transient states $\mathcal{A} \setminus (\mathcal{G} \cup B_{\mathcal{A},\mathcal{G}})$, the "bad" states $B_{\mathcal{A},\mathcal{G}} \cup (S \setminus (\mathcal{A} \cup \mathcal{G}))$ and the "goal" states $\mathcal{G}$. Therefore consider the following cases:

1. $s_0 \in \mathcal{G}$ or $s_0 \in B_{\mathcal{A},\mathcal{G}} \cup (S \setminus (\mathcal{A} \cup \mathcal{G}))$ – trivial because $Prob(s_0,\ \mathcal{A}\ \mathrm{U}\ \mathcal{G}) = 1.0$ and $Prob(s_0,\ \mathcal{A}\ \mathrm{U}\ \mathcal{G}) = 0.0$ respectively
2. $s_0 \in \mathcal{A} \setminus (\mathcal{G} \cup B_{\mathcal{A},\mathcal{G}})$ – the *c. i.* approach should be applied

The model-checking procedure for the last case is given in Algorithm 4. This algorithm is based on a sequential procedure for deriving the *c. i.* of α_g (cf. Equation (6.21) on page 124) combined with the check against the probability constraint provided by Algorithm 3 (cf. page 110).

Algorithm 4 *unboundedUntil* $(\mathbf{Q}, \mathrm{P}_{\bowtie b}(\mathcal{A}\ \mathrm{U}\ \mathcal{G}), s_0, \xi, M_{max}, \Delta_M, N_{max}, \Delta_N, \delta')$

Require: $s_0 \in \mathcal{A} \setminus (\mathcal{G} \cup B_{\mathcal{A},\mathcal{G}})$
Require: $N_{max} \in \mathbb{N}_{\geq 1}$ and $\Delta_M, M_{max} \in \mathbb{N}_{\geq 2}$
Require: Δ_M *is sufficiently large*
Require: $\xi \in \mathbb{R}_{[0.25,1.0)}$
1: $RESULT := NN$, $INVD := FALSE$
2: *Obtain:* $\mathbf{P}^B$ from $\mathbf{Q}$
3: $M := 0$, $N := 0$, $\overrightarrow{\mathbf{P}_N} := \emptyset$, $\overrightarrow{\mathbf{P}'_N} := \emptyset$, $I := 1$, $\beta := 2 \cdot \left(1 - \sqrt{\xi}\right)$
4: **repeat**
5: **if** $((I \text{ is odd}) \vee (M \geq M_{max})) \wedge (N < N_{max}) \wedge \neg INVD$ **then**
6: $N := min\{N + \Delta_N, N_{max}\}$
7: **else**
8: $M := min\{M + \Delta_M, M_{max}\}$
9: **end if**
10: $extendSamples\left(s_0, \mathbf{P}^B, \overrightarrow{\mathbf{P}_N}, \overrightarrow{\mathbf{P}'_N}, M, N\right)$
11: /*Compute: I_N and I'_N*/
12: $computeBordersUU\left(\beta, \overrightarrow{\mathbf{P}_N}, \overrightarrow{\mathbf{P}'_N}\right)$
13: $INVD := \left(I_N \cap I'_N = \emptyset\right)$
14: **if** $\neg INVD \wedge \left(A_r^{g,t}\left(\overrightarrow{\mathbf{P}'_N}\right) - A_l^g\left(\overrightarrow{\mathbf{P}_N}\right) < \delta'\right)$ **then**
15: $RESULT := checkBoundVSConfInt\left(\bowtie, b, \left[A_l^g\left(\overrightarrow{\mathbf{P}_N}\right),\ A_r^{g,t}\left(\overrightarrow{\mathbf{P}'_N}\right)\right]\right)$
16: **end if**
17: $I := I + 1$
18: **until** $((RESULT = NN) \wedge (((N < N_{max}) \wedge \neg INVD) \vee (M < M_{max})))$
19: **return** $\left(A_r^{g,t}\left(\overrightarrow{\mathbf{P}'_N}\right) - A_l^g\left(\overrightarrow{\mathbf{P}_N}\right) \geq \delta'\right)\ ?\ ERR\ :\ RESULT$

Below we discuss Algorithm 4 in more detail by analyzing its preconditions, arguments and the body. The algorithm has several arguments: $\mathbf{Q}$ – the input CTMC, $\mathrm{P}_{\bowtie b}(\mathcal{A}\ \mathrm{U}\ \mathcal{G})$ – the formula to be checked, s_0 – the initial state, ξ – the desired confidence of the result, δ' – the maximal width of the confidence interval (cf. Section 6.1),

M_{max} – the maximum number of observations in the samples, Δ_M – the number by which we increase the number of observations in the samples, N_{max} – the maximum simulation depth, Δ_N – the number of steps by which we increase the simulation depth. The preconditions of the algorithm are:

- $s_0 \in \mathcal{A} \setminus (\mathcal{G} \cup B_{\mathcal{A},\mathcal{G}})$ – the condition ensures that we are not in the trivial cases of $s_0 \in \mathcal{G}$ or $s_0 \in B_{\mathcal{A},\mathcal{G}} \cup (S \setminus (\mathcal{A} \cup \mathcal{G}))$.
- $N_{max} \in \mathbb{N}_{\geq 1}$ – because simulations start in a transient state, the simulation length should be at least one, otherwise the *c. i.* for α_g will be $[0,\ 1]$.
- $\Delta_M, M_{max} \in \mathbb{N}_{\geq 2}$ – all our results so far have been proven for $M \in \mathbb{N}_{\geq 2}$.
- Δ_M *is sufficiently large* – the minimal value of M should be large enough for the Central Limit theorem to be applicable.
- $\xi \in \mathbb{R}_{[0.25,1.0)}$ – is enforced by Equation (6.12) (cf. Table 6.2 on page 121).

Further we discuss the algorithm step by step. The initialization part takes place in lines 1–3. There, among others, we compute β from the overall confidence ξ and the embedded DTMC $\mathbf{P}^B$ where the states in $\mathcal{G}$ and $B_{\mathcal{A},\mathcal{G}} \cup (S \setminus (\mathcal{A} \cup \mathcal{G}))$ are made absorbing.

The main loop is present in lines 4–18, it does iterations until either the model-checking problem is answered or the maximum of both M and N are reached. Note that, if $M = M_{max}$ and $N < N_{max}$ then iterations are terminated if the intervals I_N and $I_N^{'}$ (cf. Equation (6.23) on page 124) do not intersect. The reason for that is that the non-intersecting intervals indicate an improper simulation run which can be cured only by increasing the number of observations (M). For more detail see Section 6.2.5.

Lines 5–9 of the main loop are devoted to increasing M and N. The condition on line 5 ensures the increase of N if the conjunction of the following conditions hold: *(i)* it is an odd iteration or the maximum number of observation is reached, *(ii)* the the maximum simulation length is not reached, *(iii)* the intervals I_N and $I_N^{'}$, obtained on the previous iteration, intersect. In all other cases we increase M.

The function *extendSamples* (...), called in line 10, depending on increase of M or N, adds more observations to the samples or simulates the observations until the new epoch N. At this line, for sufficiently large values of M we can take $\overrightarrow{\mathbf{P}_N} = \overrightarrow{\mathbf{P}'_N}$ due to Theorem 25 (cf. page 120).

In lines 12–13, we compute the *c. i.* I_N and $I_N^{'}$, and then check their validity. The result of the latter is stored in the auxiliary variable *INVD*.

The *if* clause in lines 14 to 16 is responsible for invoking Algorithm 3. Note that it can be invoked only if: *(i)* $INVD \neq TRUE$, i. e. the *c. i.* of α_g is not detected as invalid, *(ii)* the width of the derived *c. i.* is less than δ' (cf. Section 6.1).

Note that in the algorithm we bound the maximum allowed sample size and the simulation depth. This is done in order to control the simulation length and to assure the algorithm termination even if $b = \alpha_g$. Also, we do not stop iterations only because the desired width of the *c. i.* is reached. Instead, if the definite answer is not yet known, we continue simulation. The latter is done because it is expected to increase the confidence levels of the algorithm since more samples means greater accuracy.

In case M_{max} and N_{max} are set to infinity, the convergence of Algorithm 4 is *a. s.* guaranteed under the assumption that $|b - \alpha_g| = \delta > 0$. The latter is due to the strong

law of large numbers for Bernoulli trials (cf. Section 5.7) and the fact that we work with an absorbing Markov chain. In practice we have finite M_{max} and N_{max} and thus the algorithm can terminate before the desired width of the *c. i.* is reached. In the latter case, in line 19, we return the error value (*ERR*) in order to indicate a faulty simulation run.

Clearly, not taking into account the possible drop of confidence due to the naive sequential *c. i.* approach, the confidence of the algorithm's definite answer should be at least ξ, if $0 < \delta' \leq \delta$ (cf. Section 6.1).

6.3 Steady-state operator

Let us first recall the numerical model checking procedure for the steady-state operator, i.e. $\mathrm{S}_{\bowtie b}(\mathcal{G})$, on a CTMC $(S,\ \mathbf{Q},\ L)$. This procedure is fully explained in Section 1.2.1 and consists of two parts: the graph analysis, and the numerical computations. More precisely, the model-checking procedure goes as follows:

1. Using graph analysis, all the BSCCs of the CTMC are determined. This yields a set $\{B_i\}_{i \in I}$ where $I = \{1, 2, 3, \ldots, K\}$ is a set of indices, with K being the number of BSCCs, and B_i the set of states belonging to BSCC i.

2. For all $i \in I$ and any $s_i \in B_i$, the steady-sate probability $\pi_i^g = \mathit{Prob}^{\infty}(s_i,\ \mathcal{G})$ is computed. This is done by solving a system of linear equations for B_i.

3. For all $s_0 \in S$, the following steps are undertaken:

   (a) For BSCC B_i the probability to reach B_i from s_0, i.e. $p_i^{s_0} = \mathit{Prob}(s_0,\ \Diamond B_i)$, is computed by solving a system of linear equations.

   (b) The long-run probability to be in $\mathcal{G}$ state, if started in state s_0, is computed as:
   $$\mathit{Prob}^{\infty}(s_0,\ \mathcal{G}) = \sum_{i \in I} p_i^{s_0} \cdot \pi_i^g \tag{6.25}$$

   (c) The probability $\mathit{Prob}^{\infty}(s_0,\ \mathcal{G})$ is checked against the constraint $\bowtie\ b$ in order to verify the validity of the formula $\mathrm{S}_{\bowtie b}(\mathcal{G})$ in state s_0.

When using simulations, we assume that the structure of the Markov chain is known, i.e. we can distinguish between the transient states and the ones that belong to different BSCCs. Then we are able to provide *c. i.* for the probabilities $p_i^{s_0}$ and π_i^g. The former can be done using the model-checking procedure for unbounded-until operator explained in Section 6.2, and the latter using the discrete-time simulation approach discussed in Section 5.6. The desired *c. i.* for $\mathit{Prob}^{\infty}(s_0,\ \mathcal{G})$ is then constructed by Equation (6.25).

Before we move on with the details of the sketched procedure, we should note that we provide two techniques for model checking $\mathrm{S}_{\bowtie b}(\mathcal{G})$, the first one is purely based on the simulation approach as described above and the second one is a hybrid combining both numerically computed probabilities $p_i^{s_0}$ and the *c. i.* for π_i^g. The reason for having two approaches is going to be explained later.

6.3.1 The pure DES approach

In this section we derive a procedure that allows to model check $\mathrm{S}_{\bowtie b}(\mathcal{G})$ by using pure discrete event simulation. In the following, we first consider the simplest way of deriving the *c. i.* for probability $Prob^{\infty}(s_0,\, \mathcal{G})$. Then, we discuss the effectiveness of the derived *c. i.* and the possible ways for its improvement. After that, we introduce the model-checking algorithm for $\mathrm{S}_{\bowtie b}(\mathcal{G})$ and analyze its convergence and efficiency issues.

Deriving the confidence interval

For a given confidence ξ_i^s, we can compute the *c. i.* for probability π_i^g, i. e.

$$\forall i \in I : Prob\left(A_l^i \leq \pi_i^g \leq A_r^i\right) = \xi_i^s, \tag{6.26}$$

using the simulation algorithm of Section 5.6. Let us fix now an initial state $s_0 \in S$ then for a given confidence ξ_i^r the *c. i.* for probability $p_i^{s_0}$, i. e.

$$\forall i \in I : Prob\left(A_l^{s_0}\left(\overrightarrow{\mathbf{P}}_i\right) \leq p_i^{s_0} \leq A_r^{s_0}\left(\overrightarrow{\mathbf{P}'_i}\right)\right) \succeq \xi_i^r. \tag{6.27}$$

can be computed by employing the simulation algorithm of Section 6.2.

Note that we use a new *c. i.* notation, namely we omit the samples from the *c. i.* borders of π_i^g, add the upper indexes for the *c. i.* borders of π_i^g and $p_i^{s_0}$, and change the meaning of the samples lower index, which now contains the identifier of BSCC B_i. We also assume that the *c. i.* borders belong to the interval $\mathbb{R}_{[0,1]}$. This is because $\pi_i^g, p_i^{s_0} \in \mathbb{R}_{[0,1]}$ and if we, for instance, have $A_l^{s_0}\left(\overrightarrow{\mathbf{P}}_i\right) < 0$ or $A_r^i > 1$ then, without loss of generality, we can assign $A_l^{s_0}\left(\overrightarrow{\mathbf{P}}_i\right) = 0$ and $A_r^i = 1$.

In order to obtain the *c. i.* for $Prob^{\infty}(s_0,\, \mathcal{G})$ we shall combine the *c. i.* given by Equations (6.26) and (6.27) using Equation (6.25). This is done by the following theorem.

Theorem 33 *For the* c. i. *given by Equations (6.26) and (6.27), and based on independently obtained samples, the following* c. i. *results:*

$$Prob\left(\sum_{i=1}^{K} A_l^{s_0}\left(\overrightarrow{\mathbf{P}}_i\right) \cdot A_l^i \leq Prob^{\infty}(s_0,\, \mathcal{G}) \leq \sum_{i=1}^{K} A_r^{s_0}\left(\overrightarrow{\mathbf{P}'_i}\right) \cdot A_r^i\right) \succeq \prod_{i=1}^{K} \xi_i^r \xi_i^s,$$

under the assumption that for all $i \in I : A_l^{s_0}\left(\overrightarrow{\mathbf{P}}_i\right), A_r^{s_0}\left(\overrightarrow{\mathbf{P}'_i}\right), A_l^i, A_r^i \in \mathbb{R}_{[0,1]}$.

Proof See the proof of Theorem 72 from Appendix C.2. □

As a simple consequence of Theorem 33, assuming that for all $i \in I$ we have $\xi_i^r = \xi_i^s = \xi$ for some confidence ξ and $n = 2 \cdot K$, we get the following *c. i.*

$$Prob\left(\sum_{i=1}^{K} A_l^{s_0}\left(\overrightarrow{\mathbf{P}}_i\right) \cdot A_l^i \leq Prob^{\infty}(s_0,\, \mathcal{G}) \leq \sum_{i=1}^{K} A_r^{s_0}\left(\overrightarrow{\mathbf{P}'_i}\right) \cdot A_r^i\right) \succeq (\xi)^n. \tag{6.28}$$

This *c. i.* is more convenient for us, because, when given a desired overall confidence ϑ for the *c. i.* of $Prob^{\infty}(s_0,\, \mathcal{G})$, we can easily derive the value of ξ as the n'th root of ϑ.

The efficiency of the derived confidence interval

At this point it is apparent that the dependency of ξ from ϑ is very poor. Clearly, there is at least one element in the set I and, since $\vartheta \in \mathbb{R}_{[0,1)}$, the value of ξ will exceed ϑ. Remember that, the larger the confidence ξ, the wider the *c. i.* given by Equations (6.26) and (6.27). The latter implies a wider *c. i.* for $Prob^{\infty}(s_0, \mathcal{G})$. Therefore, in order to keep this *c. i.* as tight as possible, a significant increase of the sample sizes should be considered, increasing the costs of simulation.

A similar effect was discovered in [145] when using Monte Carlo simulation and hypothesis testing for model checking nested CSL formulas. In this case, the indifference intervals of sub-formulas have to be much smaller than the defined indifference region of the outer probabilistic operator, causing a significant increase of sample sizes. The solution was suggested to model check all the sub formulas using numerical techniques and then apply simulation only to the outermost operator.

In our setting, for all $i \in I$, we could consider computing either all π_i^g or all $p_i^{s_0}$ probabilities numerically, and/or use simple graph analysis in order to avoid computing the *c. i.* for some trivial cases. Both of these possibilities will reduce the value of ξ, but next we will first discuss the latter approach and the former one will be treated later.

Let $I_{\mathcal{G}} = \{i \in I \mid B_i \cap \mathcal{G} \neq \emptyset\}$, then we can automatically say that for any $i \in I \setminus I_{\mathcal{G}}$: $\pi_i^g = 0$, where the set I is assumed to be known and $I_{\mathcal{G}}$ is easy to compute. Next, for any $i \in I_{\mathcal{G}}$ let $S_i^0 \subseteq S$ be a set of states such that for any $s_0 \in S_i^0$ the BSCC B_i is unreachable from s_0, then clearly $p_i^{s_0} = 0$. As a result, we need to derive the *c. i.* of π_i^g only when $i \in I_{\mathcal{G}}^{s_0} = \left\{i \in I_{\mathcal{G}} | s_0 \notin S_i^0\right\}$. Also, for any $i \in I_{\mathcal{G}}$, let $S_i^1 \subseteq S$ be a set of states such that for any $s_0 \in S_i^1$ all the paths from s_0 lead to B_i, then clearly $p_i^{s_0} = 1$. The computation of sets S_i^0 and S_i^1 can be efficiently done using a graph-based analysis suggested in [31]. To summarize, using these observations we can avoid computing the *c. i.* for probabilities that are definitely zero or one. This, taking into consideration Equation (6.28), will reduce the confidence levels needed for Equations (6.26) and (6.27), and also reduce the number of *c. i.* that need to be computed. All in all, these optimizations will result in a tighter *c. i.* for $Prob^{\infty}(s_0, \mathcal{G})$ and possibly fewer computations.

The model-checking algorithm

The model-checking procedure is now summarized in Algorithm 5. This algorithm has parameters that are almost the same as of Algorithm 4 therefore we will only indicate the differences, namely: $\mathrm{S}_{\bowtie b}(\mathcal{G})$ – the model-checking problem and ϑ – the desired confidence of the result. Note that, for simplicity we use the same constants Δ_M, M_{max} for sample sizes when computing the *c. i.* of both $p_i^{s_0}$ and π_i^g. This is not a serious limitation and can be easily changed.

Below we discuss Algorithm 5 in more details by analyzing its preconditions and the body. At the end we also consider its convergence and efficiency.

The preconditions of the given algorithm overlap with the preconditions of Algorithm 4. We do not put any conditions on the initial state s_0 because due to the graph analysis we are not going to compute the *c. i.* for $p_i^{s_0}$ in the case of $s_0 \in B_i$. We set the condition $\vartheta \in \mathbb{R}_{[0.0625,1.0)}$, because the parameter $\xi \in \mathbb{R}_{[0.25,1.0)}$ of Algorithm 4 is computed as an n'th root of ϑ with $n \in \mathbb{N}_{\geq 2}$.

Algorithm 5 *steadyState* $(\mathbf{Q}, \mathrm{S}_{\bowtie b}(\mathcal{G}), s_0, \vartheta, M_{max}, \Delta_M, N_{max}, \Delta_N, \delta')$

Require: $N_{max} \in \mathbb{N}_{\geq 1}$ and $\Delta_M, M_{max} \in \mathbb{N}_{\geq 2}$
Require: Δ_M *is sufficiently large*
Require: $\vartheta \in \mathbb{R}_{[0.0625,1.0)}$
1: *Obtain:* $\{B_i\}_{i\in I}$, $I_{\mathcal{G}}^{s_0}$
2: **if** $I_{\mathcal{G}}^{s_0} = \emptyset$ **then**
3: **return** $(0 \bowtie b)$
4: **end if**
5: /*Use Algorithm 6 to initialize *MSCIU*, *MCIS*, *MTSU*, *MTSS*, ξ, β*/
6: $REACH := init\left(\mathbf{Q}, I_{\mathcal{G}}^{s_0}, \{B_i\}_{i\in I}, \vartheta, MSCIU, MCIS, MTSU, MTSS, \xi, \beta\right)$
7: *Obtain:* $\mathbf{P}$, $\overrightarrow{q}$ from $\mathbf{Q}$, and set $M^s := 0$, $I := 0$, $INVD := FALSE$
8: **repeat**
9: **for all** $j \in I_{\mathcal{G}}^{s_0} \bigwedge REACH$ **do**
10: /*Use Algorithm 7 to compute the *c. i.* for $p_i^{s_0}$*/
11: $MSCIU[j] := confintUU\left(s_0, \mathbf{P}^B, j, \{B_i\}_{i\in I}, MTSU, \beta, I, INVD, \ldots\right)$
12: **if** *INVD* **then**
13: **return** *NN*
14: **end if**
15: **end for**
16: /*Use Algorithm 8 to check the *c. i.* of $Prob^{\infty}(s_0, \mathcal{G})$ against $\bowtie b$*/
17: $RESULT := checkCI\left(I_{\mathcal{G}}^{s_0}, \bowtie, b, MSCIU, MCIS, \delta'\right)$
18: **if** $RESULT = NN$ **then**
19: $inc\left(M^s, \Delta_M, M_{max}\right)$
20: **for all** $i \in I_{\mathcal{G}}^{s_0}$ **do**
21: /*Use algorithm of Section 5.6 to compute the *c. i.* for π_i^g*/
22: $MCIS[i] := confintSS\left(\mathbf{P}, \overrightarrow{q}, B_i, MTSS[i], M^s, \xi\right)$
23: **end for**
24: **end if**
25: /*Use Algorithm 8 to check the *c. i.* of $Prob^{\infty}(s_0, \mathcal{G})$ against $\bowtie b$*/
26: $RESULT := checkCI\left(\bowtie, b, MSCIU, MCIS, \delta'\right)$
27: $I := I + 1$
28: **until** $((RESULT = NN) \wedge moreIter\left(MTSU, M^s, N_{max}, M_{max}\right))$
29: **return** $isCITight\left(MSCIU, MCIS, \delta'\right)\ ?\ RESULT\ :\ ERR$

The line 1 of the algorithm is devoted for computing the sets $\{B_i\}_{i\in I}$ and $I_{\mathcal{G}}^{s_0}$. The former set can be computed outside the algorithm, along with the auxiliary sets $I_{\mathcal{G}}$ and $\left\{S_i^0\right\}_{i\in I_{\mathcal{G}}}$, since they are constant for the given CTMC and the model-checking problem. The set $I_{\mathcal{G}}^{s_0}$ is a set of BSCC indexes such that these BSCCs are reachable from s_0 and contain $\mathcal{G}$ states. It depends on the initial state s_0 and therefore has to be computed inside the algorithm.

Lines 2 to 6 do the initialization part of the algorithm. If $I_{\mathcal{G}}^{s_0}$ is empty then there is nothing to solve because $Prob^{\infty}(s_0, \mathcal{G}) = 0$ and thus we can give the answer right away, otherwise the function:

$$init\left(\mathbf{Q}, I_{\mathcal{G}}^{s_0}, \{B_i\}_{i\in I}, \vartheta, MSCIU, MCIS, MTSU, MTSS, \xi, \beta\right)$$

is called. This function is described by Algorithm 6 and its tasks are as follows:

- To initialize the required maps that map BSCC indexes $i \in I_{\mathcal{G}}^{s_0}$ to the following data:

Algorithm 6 *init* $\left(\mathbf{Q}, I_{\mathcal{G}}^{s_0}, \{B_i\}_{i\in I}, \vartheta, MSCIU, MCIS, MTSU, MTSS, \xi, \beta\right)$

1: *Obtain:* $\left\{S_i^1\right\}_{i\in I_{\mathcal{G}}}$ using $\mathbf{Q}$ and $\{B_i\}_{i\in I}$
2: **if** $\exists i \in I_{\mathcal{G}}^{s_0}$: $s_0 \in S_i^1$ **then**
3: $REACH := FALSE$
4: $\xi := \vartheta$
5: *Allocate:* $MCIS$, $MTSS$ – maps of 1 element each
6: $MSCIU[i] := [1, 1]$
7: **else**
8: $REACH := TRUE$
9: $R = |I_{\mathcal{G}}^{s_0}|$, $n := 2 \cdot R$, $\xi := \sqrt[n]{\vartheta}$, $\beta := 2 \cdot \left(1 - \sqrt{\xi}\right)$
10: *Allocate:* $MSCIU$, $MCIS$, $MTSU$, $MTSS$ – maps of R elements each
11: **end if**
12: **for all** $i \in I_{\mathcal{G}}^{s_0}$ **do**
13: $MCIS[i] := [0, 1]$
14: **end for**
15: **return** $REACH$

$MTSU$ – $MTSU[i]$ stores a tuple $\left(M, N, \overrightarrow{\mathbf{P}_N}, \overrightarrow{\mathbf{P}'_N}\right)$ for computing the *c. i.* of $p_i^{s_0}$

$MTSS$ – $MTSS[i]$ stores a tuple $\left(s_i, \overrightarrow{S}, \overrightarrow{T}\right)$ for computing the *c. i.* of π_i^g with $s_i \in B_i$ being a chosen regeneration point and $\overrightarrow{S}$ with $\overrightarrow{T}$ being the required sample vectors, see Section 5.6.

$MSCIU$ – $MSCIU[i]$ stores the *c. i.* of $p_i^{s_0}$

$MCIS$ – $MCIS[i]$ stores the *c. i.* of π_i^g

- To compute the values of ξ, β that are implicit results of this algorithm. Here ξ is the confidence for computing the *c. i.* of π_i^g and β defines the confidence for computing the *c. i.* of $p_i^{s_0}$, see lines 9 and 4.
- To initialize the *c. i.* of π_i^g with $[0, 1]$ that is a technical step, see line 12 to 14.
- To sort out the trivial case when all paths from s_0 go to some BSCC B_i with $i \in I_{\mathcal{G}}^{s_0}$. In this case, see line 2 to 7, it is known that $p_i^{s_0} = 1$ and therefore we initialize $MSCIU[i]$ with $\{[1, 1]\}$, also we have to do simulation only to compute the *c. i.* of π_i^g and therefore we set $\xi = \vartheta$.
- To return an explicit result *TRUE* if we have to use simulation to compute the *c. i.* for some $p_i^{s_0}$, i.e. it is not a trivial case, and *FALSE* otherwise, see lines 3 and 8.

Note that, similar to $\left\{S_i^0\right\}_{i\in I_{\mathcal{G}}}$, the set $\left\{S_i^1\right\}_{i\in I_{\mathcal{G}}}$ can be reused.

Back to Algorithm 5, on line 7 we obtain, possibly precomputed, $\mathbf{P}$ – the embedded DTMC of $\mathbf{Q}$, and $\overrightarrow{q}$ - the vector of exit rates for the matrix $\mathbf{Q}$. We also initialize the regeneration-cycle counter M^s which we are going to use for all *c. i.* of π_i^g. Lines 8 to 28 form the main loop of the algorithm, this cycle iterates until either we have a definite answer to the model checking problem or we have reached both the maximum number

of regeneration cycles when computing the *c. i.* of π_i^g and the maximum number of observations and simulation depth in the samples needed for computing $p_i^{s_0}$ for all $i \in I_{\mathcal{G}}^{s_0}$. The check for the latter is done using function *moreIter* $(MTSU, M^s, N_{max}, M_{max})$.

Algorithm 7 *confintUU* $\left(s_0, \mathbf{P}^B, j, \{B_i\}_{i\in I}, MTSU, \beta, I, INVD, \ldots\right)$

Obtain: $\mathbf{P}^B$ using $\mathcal{G} := B_j$, $B_{\mathcal{A},\mathcal{G}} := \bigcup_{i\in I\setminus\{j\}} B_i$ and $\mathcal{I} := \emptyset$
$\left(M, N, \overrightarrow{\mathbf{P}_N}, \overrightarrow{\mathbf{P}'_N}\right) := MTSU[j]$
if $((I \text{ is odd}) \vee ((M \geq M_{max}))) \wedge (N < N_{max})$ **then**
  $N := min\{N + \Delta_N, N_{max}\}$
else
  $M := min\{M + \Delta_M, M_{max}\}$
end if
extendSamples $\left(s_0, \mathbf{P}^B, \overrightarrow{\mathbf{P}_N}, \overrightarrow{\mathbf{P}'_N}, M, N\right)$
/*Compute: I_N and I'_N*/
computeBordersUU $\left(\beta, \overrightarrow{\mathbf{P}_N}, \overrightarrow{\mathbf{P}'_N}\right)$
while $\left(INVD := \left(I_N \cap I'_N = \emptyset\right)\right) \wedge (M < M_{max})$ **do**
  $M := min\{M + \Delta_M, M_{max}\}$
  extendSamples $\left(s_0, \mathbf{P}^B, \overrightarrow{\mathbf{P}_N}, \overrightarrow{\mathbf{P}'_N}, M, N\right)$
  /*Compute: I_N and I'_N*/
  computeBordersUU $\left(\beta, \overrightarrow{\mathbf{P}_N}, \overrightarrow{\mathbf{P}'_N}\right)$
end while
return $\left[A_l^g\left(\overrightarrow{\mathbf{P}_N}\right),\ A_r^{g,t}\left(\overrightarrow{\mathbf{P}'_N}\right)\right]$

In the beginning of the main cycle, line 9 to 15, for every $j \in I_{\mathcal{G}}^{s_0}$, the *c. i.* of $p_j^{s_0}$ is recomputed if $REACH = TRUE$, i.e. it is not a trivial case, by invoking:

$$confintUU\left(s_0, \mathbf{P}^B, j, \{B_i\}_{i\in I}, MTSU, \beta, I, INVD, \ldots\right)$$

For brevity, we omitted some of the call parameters such as: M_{max}, Δ_M, N_{max}, Δ_N.

Function *confintUU* $(\ldots)$ is described by Algorithm 7. It computes the *c. i.* of $p_j^{s_0}$ by extending the sample size or the simulation depth. This function is based on the main loop of Algorithm 4 and in case the invalid intervals I_N and I'_N are detected it tries to cure the problem by increasing the sample size (cf. Section 6.2.5). In the latter case, if the maximum sample size is reached then there is no hope of producing the correct *c. i.* of $Prob^{\infty}(s_0, \mathcal{G})$ and thus Algorithm 5 terminates (cf. lines 12 to 14).

Next, on line 17, function *checkCI* $\left(I_{\mathcal{G}}^{s_0}, \bowtie, b, MSCIU, MCIS, \delta'\right)$ is called. The behavior of this function is given by Algorithm 8 which main task is to compute and check the current *c. i.* of $Prob^{\infty}(s_0, \mathcal{G})$ against the probability constraint $\bowtie b$. The maps *MSCIU* and *MCIS* are among the arguments of the procedure because *MSCIU*$[i]$ contains the *c. i.* for $p_i^{s_0}$ and *MCIS*$[i]$ the *c. i.* for π_i^g. Note that operations $+$ and $*$ on the *c. i.* are done component wise. After the *c. i.* of $Prob^{\infty}(s_0, \mathcal{G})$ is ready and it's width is less than δ', Algorithm 3 is invoked to test the interval against the constraint $\bowtie b$. The result is returned by function *checkCI* $(\ldots)$. Note that if the width of the obtained *c. i.* is not less than δ' the algorithm returns *NN* notifying Algorithm 5 that more simulations is needed.

Algorithm 8 *checkCI* $\left(I_{\mathcal{G}}^{s_0}, \bowtie, b, MSCIU, MCIS, \delta'\right)$

```
CINT := [0, 0]
for all i ∈ I_G^{s0} do
  CINT := CINT + MSCIU[i] * MCIS[i]
end for
if width (CINT) < δ' then
  /*Use Algorithm 3 to answer the model-checking problem*/
  return checkBoundVSConfInt (⋈, b, CINT)
else
  return NN
end if
```

In case we still do not have a definite answer to the model-checking problem, lines 18 to 24 of Algorithm 5, we increase the number of regeneration cycles we want to consider and after doing simulations recompute the *c. i.* of all π_i^g with $i \in I_{\mathcal{G}}^{s_0}$. Simulations and computations of the *c. i.* are done within function *confintSS* $(\mathbf{P}, \overrightarrow{q}, B_i, MTSS[i], M, \xi)$. We do not provide pseudo code for it because its functionality is a straight-forward consequence of the procedure given in Section 5.6.

At the end of the main cycle (line 26) we recompute the *c. i.* of $Prob^{\infty}(s_0, \mathcal{G})$ and check it against the constraint $\bowtie b$ using function *checkCI* (...).

Note that, since we bound the maximum sample size and simulation depth (like it was done in Algorithm 4 on page 130), in line 29 we test whether the derived *c. i.* is tight enough, and if it is not then we return the error value.

Convergence, confidence and efficiency

The *c. i.* of $Prob^{\infty}(s_0, \mathcal{G})$ is based on a linear combination of *c. i.* for $p_i^{s_0}$ and π_i^g. Therefore, in case of no bounds on the simulation depth N, the sample size M, and $Prob^{\infty}(s_0, \mathcal{G}) \neq b$, the *a. s.* convergence of Algorithm 5 is ensured by the *a. s.* convergence of the *c. i.* for $p_i^{s_0}$ (cf. Section 6.2.6) and π_i^g.

The main cycle of Algorithm 5 has the termination conditions similar to the ones of the Algorithm 4. Also, the same Algorithm 3 is used for comparing the *c. i.* to the probability constraint. This implies that the confidence level for the definite answer of the algorithm is at least ϑ if $0 < \delta' \leq |Prob^{\infty}(s_0, \mathcal{G}) - b|$.

To discuss the efficiency of Algorithm 5 we provide some empirical observations. Notice that the dependency of confidence ξ from the overall confidence ϑ, although improved, is still rather poor, namely ξ is the n'th root of ϑ where $n := 2 \cdot R$ and $R = |I_{\mathcal{G}}^{s_0}|$. The procedure of computing the *c. i.* of $Prob^{\infty}(s_0, \mathcal{G})$ involves producing two independent samples for the unbounded-reachability problem which may result in an invalid *c. i.* for $p_i^{s_0}$ (cf. Algorithm 7). All this makes the suggested procedure computationally heavy. Among other potential inefficiencies is that for different $s_0 \in S$ the *c. i.* for $p_i^{s_0}$ requires separate simulations whereas the *c. i.* and samples for π_i^g can be reused. The last reason for improving the provided algorithm is that the memory needed for storing all required samples for $p_i^{s_0}$ and efforts for storing or recomputing $\mathbf{P}^B$ matrices (cf. Algorithm 7) can be quite substantial.

The solution for the issues above can be as simple as computing probabilities $p_i^{s_0}$

numerically. This allows, for any fixed $i \in I$, to compute the reachability probabilities $p_i^{s_0}$ for all $s_0 \in S$ at once with a predefined error bound ε. In terms of Algorithm 5, it will exclude the use of the maps *MSCIU* and *MTSU*, significantly simplify functions *init* (...), *checkCI* (...) and *moteIter* (...), and remove function *confintUU* (...). Also, it will reduce the value of confidence ξ, making it an n'th root of ϑ with $n := R$. The next subsection presents theory and model-checking algorithms for the outlined hybrid approach.

6.3.2 The hybrid approach

This subsection presents a hybrid approach for model checking the $\mathrm{S}_{\bowtie b}(\mathcal{G})$ operator, based on discrete event simulation and numerical computations. Following the ideas of Section 6.3.1, we intend to optimize the model-checking procedures by computing reachability probabilities $p_i^{s_0}$ using the well-known numerical approach. In the following, we first consider the way of deriving a "hybrid" *c. i.* for probability $Prob^{\infty}(s_0,\ \mathcal{G})$. This interval will combine *c. i.* of the steady-state probabilities π_i^g with the numerical error bounds of reachability probabilities $p_i^{s_0}$. After that, we introduce the model-checking algorithm for $\mathrm{S}_{\bowtie b}(\mathcal{G})$ and compare it to the model-checking algorithm devised in Section 6.3.1.

Deriving the "hybrid" confidence interval

For all $i \in I$, $s_0 \in S$ and errors ε_i, let us compute the bounds for $p_i^{s_0}$ numerically, with the help of the approach discussed in Section 1.2.1. This yields:

$$\forall i \in I\ :\ \widetilde{p}_i^{s_0} - \varepsilon_i \le p_i^{s_0} \le \widetilde{p}_i^{s_0} + \varepsilon_i, \tag{6.29}$$

where $\widetilde{p}_i^{s_0}$ is the probability computed numerically. For any $i \in I$, and a given confidence ξ_i^s, as before, we have the *c. i.* of π_i^g given by Equation (6.26).

Now under the assumption that for all $i \in I$ we have $\widetilde{p}_i^{s_0} - \varepsilon_i$, $\widetilde{p}_i^{s_0} + \varepsilon_i$, A_l^i and A_r^i in $\mathbb{R}_{[0,1]}$, using Equation (6.25), we can provide a *c. i.* for $Prob^{\infty}(s_0,\ \mathcal{G})$ in the form of the following theorem.

Theorem 34 (The *c. i.* of the error) *For the* c. i. *given by Equation (6.26), based on independently obtained samples, and the error bounds given by Equation (6.29), the following* c. i. *results:*

$$Prob\left(\sum_{i=1}^{K}(\widetilde{p}_i^{s_0} - \varepsilon_i)\cdot A_l^i \le Prob^{\infty}(s_0,\ \mathcal{G}) \le \sum_{i=1}^{K}(\widetilde{p}_i^{s_0} + \varepsilon_i)\cdot A_r^i\right) \succeq \prod_{i=1}^{K}\xi_i^s,$$

under the assumption that for all $i \in I : \widetilde{p}_i^{s_0} - \varepsilon_i,\ \widetilde{p}_i^{s_0} + \varepsilon_i,\ A_l^i, A_r^i \in \mathbb{R}_{[0,1]}$.

Proof See the proof of Theorem 74 from Appendix C.2. □

As before, taking $\varepsilon_i = \varepsilon$ and $\xi_i^s = \xi$ for all $i \in I$ yields:

$$Prob\left(\sum_{i=1}^{K}(\widetilde{p}_i^{s_0} - \varepsilon)\cdot A_l^i \le Prob^{\infty}(s_0,\ \mathcal{G}) \le \sum_{i=1}^{K}(\widetilde{p}_i^{s_0} + \varepsilon)\cdot A_r^i\right) \succeq (\xi)^n,\ \text{with } n = K. \tag{6.30}$$

In practice, for any given s_0 it suffices to compute the probability bounds of $p_i^{s_0}$ for all $i \in I_{\mathcal{G}}$ and the *c. i.* of π_i^g for all $i \in I_{\mathcal{G}}^{s_0}$, because for all $i \in I \setminus I_{\mathcal{G}}^{s_0}$ we have $\pi_i^g = 0$. This reduces the number of numerical computations and the simulation effort. The latter is due to computing ξ as an n'th root of the overall confidence ϑ with a possibly smaller $n = |I_{\mathcal{G}}^{s_0}|$.

Algorithm 9 *steadyStateHybrid* $(\mathbf{Q}, \mathrm{S}_{\bowtie b}(\mathcal{G}), s_0, \vartheta, \varepsilon, M_{max}, \Delta_M, \delta')$

Require: $\Delta_M, M_{max} \in \mathbb{N}_{\geq 2}$
Require: Δ_M *is sufficiently large*
1: *Obtain:* $\{B_i\}_{i\in I}$, $\left\{\{p_i^{s_0}\}_{s_0\in S}\right\}_{i\in I_{\mathcal{G}}}$, $I_{\mathcal{G}}^{s_0}$
2: **if** $I_{\mathcal{G}}^{s_0} = \emptyset$ **then**
3: **return** $(0 \bowtie b)$
4: **end if**
5: /*Use Algorithm 10 to initialize *MCIS*, *MTSS*, ξ*/
6: *initHybrid* $\left(I_{\mathcal{G}}^{s_0}, \vartheta, MCIS, MTSS, \xi\right)$
7: *Obtain:* $\mathbf{P}$, $\overrightarrow{q}$ from $\mathbf{Q}$, and set $M^s := 0$
8: /*Use Algorithm 11 to check the *c. i.* of $Prob^{\infty}(s_0, \mathcal{G})$ against $\bowtie b$*/
9: $RESULT :=$ *checkCIHybrid* $\left(I_{\mathcal{G}}^{s_0}, \bowtie, b, \varepsilon, \left\{\{p_i^{s_0}\}_{s_0\in S}\right\}_{i\in I_{\mathcal{G}}}, MCIS, \delta'\right)$
10: **while** $((RESULT = NN) \bigwedge (M^s < M_{max}))$ **do**
11: *inc* (M^s, Δ_M, M_{max})
12: **for all** $i \in I_{\mathcal{G}}^{s_0}$ **do**
13: /*Use algorithm of Section 5.6 to compute the *c. i.* for π_i^g*/
14: $MCIS[i] :=$ *confintSS* $(\mathbf{P}, \overrightarrow{q}, B_i, MTSS[i], M^s, \xi)$
15: **end for**
16: /*Use Algorithm 11 to check the *c. i.* of $Prob^{\infty}(s_0, \mathcal{G})$ against $\bowtie b$*/
17: $RESULT :=$ *checkCIHybrid* $\left(\bowtie, b, \varepsilon, \left\{\{p_i^{s_0}\}_{s_0\in S}\right\}_{i\in I_{\mathcal{G}}}, MCIS, \delta'\right)$
18: **end while**
19: **return** *isCITight* $\left(\varepsilon, \left\{\{p_i^{s_0}\}_{s_0\in S}\right\}_{i\in I_{\mathcal{G}}}, MCIS, \delta'\right)$? $RESULT$: ERR

The model-checking algorithm

The hybrid approach is now summarized in the form of Algorithm 9. Below we discuss this algorithm in more detail by explaining the changes in its arguments, preconditions and the body, when compared to Algorithm 5.

First, notice that the parameters related only to the computations of the *c. i.* for $p_i^{s_0}$ disappeared, namely: N_{max} and Δ_N. Instead there is one new parameter ε that defines the error bound for the numerical computations of probabilities $p_i^{s_0}$. The preconditions are a subset of preconditions from Algorithm 5.

In the algorithm's body (line 1) in addition to obtaining $\{B_i\}_{i\in I}$ and $I_{\mathcal{G}}^{s_0}$ we get probabilities $\left\{\{p_i^{s_0}\}_{s_0\in S}\right\}_{i\in I_{\mathcal{G}}}$ that should be computed with the error bound ε, note that the auxiliary set $\left\{S_i^0\right\}_{i\in I_{\mathcal{G}}}$ does not have to be computed since we can define $I_{\mathcal{G}}^{s_0} = \{i \in I_{\mathcal{G}} | p_i^{s_0} \neq 0\}$.

In lines 2 to 6 the initialization part takes place. The difference here is that for a

non-trivial case of $I_{\mathcal{G}}^{s_0} \neq \emptyset$, instead of calling function $init\,(\ldots)$ (cf. Algorithm 6), we invoke function:

$$initHybrid\left(I_{\mathcal{G}}^{s_0}, \vartheta, MCIS, MTSS, \xi\right)$$

given by Algorithm 10. This function is just a simplified version of $init\,(\ldots)$ where we only initialize maps $MCIS$, $MTSS$ and the confidence ξ.

Further, see line 9, prior to the main loop of the algorithm, we call function:

$$checkCIHybrid\left(I_{\mathcal{G}}^{s_0}, \bowtie, b, \varepsilon, \left\{\left\{p_i^{s_0}\right\}_{s_0 \in S}\right\}_{i \in I_{\mathcal{G}}}, MCIS, \delta'\right),$$

given by Algorithm 11. The latter one is again a simplified version of $checkCI\,(\ldots)$, provided by Algorithm 8. The main difference here is that in order to construct the *c. i.* of $Prob^{\infty}\,(s_0,\ \mathcal{G})$ we use the exact probability bounds for each $p_i^{s_0}$, this procedure is based on Equation (6.30).

The main cycle of Algorithm 9, see lines 10 to 18, has two stopping conditions, namely: *(i)* the result of the model-checking problem is definite, *(ii)* we have reached the maximum number of allowed regeneration cycles. Also the body of the main loop contains no computations for the *c. i.* of $p_i^{s_0}$.

To conclude, we should note that the hybrid algorithm looks much simpler than the one purely based on simulations. Among other improvements are the smaller values of confidence ξ, and the fact that with a minor modification Algorithm 9 can be extended to model check formula $\mathrm{S}_{\bowtie b}\,(\mathcal{G})$ for all initial states $s_0 \in S$ by reusing probability bounds for $p_i^{s_0}$ along with the samples and the *c. i.* for π_i^g.

6.4 Time-interval until operator

In this section we discuss a model-checking approach for the time-interval until operator $\mathrm{P}_{\bowtie b}\left(\mathcal{A}\ \mathrm{U}^{[t_l,t_r]}\ \mathcal{G}\right)$, with $t_l, t_r \in \mathbb{R}_{\geq 0}$ and $t_l \leq t_r$. As opposed to the operators $\mathrm{P}_{\bowtie b}\,(\mathcal{A}\ \mathrm{U}\ \mathcal{G})$ and $\mathrm{S}_{\bowtie b}\,(\mathcal{G})$, we do not simulate the embedded DTMC, but rather use a continuous-time simulation technique. Our algorithm is based on computing the sequential *c. i.* for the probability $Prob\left(s_0,\ \mathcal{A}\ \mathrm{U}^{[t_l,t_r]}\ \mathcal{G}\right)$ using terminating simulation (cf. Section 5.4) and then checking the resulting interval against the probability constraint $\bowtie\ b$ by means of Algorithm 3 (cf. page 110).

The rest of the section is organized as follows. First, we devise a simulation procedure (based on terminating simulation) that allows to determine the *c. i.* of $Prob\left(s_0,\ \mathcal{A}\ \mathrm{U}^{[t_l,t_r]}\ \mathcal{G}\right)$. For this, we recall the notion of a path in the CTMC $(S,\ \mathbf{Q},\ L)$. Then we formally define a condition of satisfying the formula $\mathcal{A}\ \mathrm{U}^{[t_l,t_r]}\ \mathcal{G}$ on a path in terms of the path states and their arrival- and departure-times. The latter will result in

Algorithm 10 $initHybrid\left(I_{\mathcal{G}}^{s_0}, \vartheta, MCIS, MTSS, \xi\right)$

1: $n := |I_{\mathcal{G}}^{s_0}|$, $\xi := \sqrt[n]{\vartheta}$
2: *Allocate:* $MCIS$, $MTSS$ – maps of n elements each
3: **for all** $i \in I_{\mathcal{G}}^{s_0}$ **do**
4: $\quad MCIS[i] := [0, 1]$
5: **end for**

Algorithm 11 *checkCIHybrid* $\left(I_{\mathcal{G}}^{s_0}, \bowtie, b, \varepsilon, \left\{\{p_i^{s_0}\}_{s_0 \in S}\right\}_{i \in I_{\mathcal{G}}}, MCIS, \delta'\right)$

$CINT := [0, 0]$
for all $i \in I_{\mathcal{G}}^{s_0}$ **do**
  $CINT := CINT + [p_i^{s_0} - \varepsilon,\ p_i^{s_0} + \varepsilon] * MCIS[i]$
end for
if *width* $(CINT) < \delta'$ **then**
  /*Use Algorithm 3 to answer the model-checking problem*/
  return *checkBoundVSConfInt* $(\bowtie, b, CINT)$
else
  return *NN*
end if

stopping criteria for the terminating simulation. Second, we summarize the complete model-checking procedure in the form of an algorithm.

The simulation procedure and the confidence interval

Recall (cf. Section 1.1.2) that a CTMC can be represented as a composition of an embedded DTMC $\mathbf{P}$ that gives the probability distributions for moving from one state to another, and for each state $s \in S$ an exponentially-distributed holding time t^s with rate $q_s = -q_{s,s}$, where $q_{s,s}$ is a diagonal element of the generator matrix $\mathbf{Q}$. A path $\sigma \in Path^{\mathcal{C}}$ in the CTMC is a sequence $\sigma = s_0\, t_0\, s_1\, t_1 \ldots$ with $\mathbf{P}(s_i, s_{i+1}) > 0$ and $t_i \in \mathbb{R}_{>0}$ for all $i \in \mathbb{N}$. The time stamps t_i denote the amount of time spent in state s_i. Considering the path σ, let us define the arrival time $t_{s_i}^a = \sum_{j=0}^{i-1} t_j$ and the departure time $t_{s_i}^d = \sum_{j=0}^{i} t_j$ for the state s_i. Note that at the time $t_{s_i}^d$ the transition to sate s_{i+1} occurs and therefore $\sigma@t = s_i$ **iff** $t \in \left[t_{s_i}^a,\ t_{s_i}^d\right)$.

The path σ satisfies the formula $\mathcal{A}\ \mathrm{U}^{[t_l, t_r]}\ \mathcal{G}$ **iff** there exists $t' \in [t_l,\ t_r]$ such that $\sigma@t' \in \mathcal{G}$ and for all $t < t'$ we have $\sigma@t \in \mathcal{A}$. The latter is **iff** there exists $s_i \in \sigma$ such that $s_i \in \mathcal{G}$ and $\left(t_{s_i}^a \leq t_r\right) \wedge \left(t_l < t_{s_i}^d\right)$ (we are in one of the $\mathcal{G}$ states within the time interval $[t_l,\ t_r]$), and if $t_{s_i}^a < t_l$ then for all $j \leq i : s_j \in \mathcal{A}$ or else for all $j < i : s_j \in \mathcal{A}$ (for all time points before t_l we reside in one of the $\mathcal{A}$ states).

In the following we are going to estimate the probability $Prob\left(s_0,\ \mathcal{A}\ \mathrm{U}^{[t_l, t_r]}\ \mathcal{G}\right)$ using terminating simulation. For that we need to simulate a set of paths in the CTMC and estimate the proportion of thereof satisfying the formula $\mathcal{A}\ \mathrm{U}^{[t_l, t_r]}\ \mathcal{G}$. The stopping criteria for the state-by-state path generation with the last generated state s_i are as follows:

A. $\left(t_{s_i}^a < t_l\right) \wedge \left(s_i \in S \setminus \mathcal{A}\right)$

B. $\left(t_{s_i}^a \leq t_r\right) \wedge \left(t_l < t_{s_i}^d\right) \wedge \left(s_i \in S \setminus (\mathcal{A} \cup \mathcal{G})\right)$

C. $\left(t_{s_i}^a \leq t_r\right) \wedge \left(t_l < t_{s_i}^d\right) \wedge \left(s_i \in \mathcal{G}\right)$

D. $\left(t_{s_i}^d > t_r\right)$

These termination conditions are complementary and their meaning is as follows:

A. – a not $\mathcal{A}$ state is visited before the time t_l.

B. – a not $\mathcal{A}$ and not $\mathcal{G}$ state is visited within the time interval $[t_l,\ t_r]$.

C. – a $\mathcal{G}$ state is visited within the time interval $[t_l,\ t_r]$.

D. – the path goes beyond the target time frame.

Let us have M paths among which Γ^g paths satisfy the given path formula. Then $\frac{\Gamma^g}{M}$ is the *p. e.* of the probability $Prob\left(s_0,\ \mathcal{A}\ \mathrm{U}^{[t_l,t_r]}\ \mathcal{G}\right)$ and for a given confidence ξ we have the following *c. i.*

$$Prob\left(A_l^g\left(\Gamma^g\right) \leq Prob\left(s_0,\ \mathcal{A}\ \mathrm{U}^{[t_l,t_r]}\ \mathcal{G}\right) \leq A_r^g\left(\Gamma^g\right)\right) \approx \xi, \tag{6.31}$$

where, since every path either satisfies the formula or not, we are in the settings of Bernoulli trials and the Agresti-Coull *c. i.* borders can be used (cf Sections 5.7 and 6.2.2).

Now, when Equation (6.31) gives the *c. i.* of $Prob\left(s_0,\ \mathcal{A}\ \mathrm{U}^{[t_l,t_r]}\ \mathcal{G}\right)$ and the stopping conditions for terminating simulation are known, we can provide a complete model-checking procedure.

Algorithm 12 *intervalUntil* $\left(\mathbf{Q}, \mathrm{P}_{\bowtie b}\left(\mathcal{A}\ \mathrm{U}^{[t_l,t_r]}\ \mathcal{G}\right), s_0, \xi, M_{max}, \Delta_M, \delta'\right)$

Require: $\Delta_M, M_{max} \in \mathbb{N}_{\geq 2}$
Require: Δ_M *is sufficiently large*
Require: $\xi \in \mathbb{R}_{[0,1.0)}$
1: *Obtain:* $\mathbf{P}$, $\overrightarrow{q}$ from $\mathbf{Q}$, and $\mathcal{I} = S \setminus (\mathcal{A} \cup \mathcal{G})$
2: $\overrightarrow{\mathbf{Q}} := (0,\ 0)$, $M := 0$, $\beta := 1 - \xi$
3: **repeat**
4: $M := min\left\{M + \Delta_M, M_{max}\right\}$
5: /*Use Algorithm 13 to extend the sample $\overrightarrow{\mathbf{Q}}$*/
6: *extendSample* $\left(s_0, \mathcal{I}, \mathcal{G}, t_l, t_r, \mathbf{P}, \overrightarrow{q}, \overrightarrow{\mathbf{Q}}, M\right)$
7: /*Compute: $A_l^g\left(\Gamma^g\right)$, $A_r^g\left(\Gamma^g\right)$*/
8: $CINT :=$ *computeBordersIU* $\left(\beta, \overrightarrow{\mathbf{Q}}\right)$
9: /*Use Algorithm 3 to answer the model-checking problem*/
10: **if** $width\left(CINT\right) < \delta'$ **then**
11: $RESULT :=$ *checkBoundVSConfInt* $\left(\bowtie, b, CINT\right)$
12: **end if**
13: **until** $\left(\left(RESULT = NN\right) \bigwedge \left(M < M_{max}\right)\right)$
14: **return** $\left(width\left(CINT\right) \geq \delta'\right)\ ?\ ERR :\ RESULT$

The model-checking algorithm

The approach to model-checking $\mathrm{P}_{\bowtie b}\left(\mathcal{A}\ \mathrm{U}^{[t_l,t_r]}\ \mathcal{G}\right)$ is summarized in Algorithm 12. Below we discuss this algorithm in detail by explaining its arguments, preconditions and the body.

The parameters of Algorithm 12 are mostly typical to the model-checking approach discussed in this chapter and have been described in the previous sections. The new parameter are: $\mathrm{P}_{\bowtie b}\left(\mathcal{A}\ \mathrm{U}^{[t_l,t_r]}\ \mathcal{G}\right)$ – the model-checking problem, ξ – the desired confidence of the result.

The preconditions on Δ_M and M_{max} are the same as in Algorithm 5 and on the confidence ξ is trivial. Further we describe the algorithm line wise.

In line 1, using the generator matrix $\mathbf{Q}$, we obtain the vector of exit rates $\overrightarrow{q}$, the embedded DTMC $\mathbf{P}$ and the set of illegal states $\mathcal{I}$. Note that they all can be stored and reused when model checking the same problem for another initial state. Next, in line 2 we perform some simple variable initialization. This includes initializing the sample $\overrightarrow{\mathbf{Q}}$ which has two components: first, the number of sampled paths, and second, the number of thereof satisfying the path formula $\mathcal{A}\ \mathrm{U}^{[t_l,t_r]}\ \mathcal{G}$. The main cycle of the algorithm takes place in lines 3 to 13, it iterates until we either get a definite answer to the model-checking problem or reach the maximum sample size.

Algorithm 13 *extendSample* $\left(s_0, \mathcal{I}, \mathcal{G}, t_l, t_r, \mathbf{P}, \overrightarrow{q}, \overrightarrow{\mathbf{Q}}, M_n\right)$

```
1: (M, Γ^g) := Q⃗
2: for j := M; j < M_n; inc(i) do
3:    s := s_0, t_s^a := 0, t_s^d := simulateTime(q_s)
4:    while (t_s^d ≤ t_l) ∧ (s ∈ A) do
5:       s := simulateState(s, P)
6:       t_s^a := t_s^d, t_s^d := t_s^d + simulateTime(q_s)
7:    end while
8:    if (t_s^a = t_l) ∨ (s ∈ A) then
9:       while (s ∉ I) ∧ (s ∉ G) ∧ (t_s^d ≤ t_r) do
10:         s := simulateState(s, P)
11:         t_s^d := t_s^d + simulateTime(q_s)
12:      end while
13:      if s ∈ G then
14:         Γ^g := Γ^g + 1
15:      end if
16:   end if
17: end for
18: Q⃗ := (M_n, Γ^g)
```

Inside the main loop we first increment the number of observations, see line 4, and then in line 6 call function *extendSample* $\left(s_0, \mathcal{I}, \mathcal{G}, t_l, t_r, \mathbf{P}, \overrightarrow{q}, \overrightarrow{\mathbf{Q}}, M_n\right)$ given by Algorithm 13. The main purpose of this function is to perform terminating simulations. As a result, the function provides the extended sample $\overrightarrow{\mathbf{Q}}$.

Let us discuss Algorithm 13 in a little more detail. Its first line is dedicated to obtaining the current sample size M and the number of successful paths Γ^g. Lines from 2 to 17 are devoted to the sample-path generation. First, in line 3 the initial state is taken to be s_0, the arrival time is set zero and the departure time is simulated. Further we split the path-generation procedure into the following two simulation parts:

1. Lines 4–7: until the time t_l checking for the termination condition *A*.
2. Lines 9–12: until the time t_r checking for the termination conditions *B.* – *D.*

It is important to note that the conditional operator on lines 8–16 ensures that the second part of the simulation procedure is invoked only if the termination condition *A.* is not satisfied. The conditional operator on lines 13–15 checks if the path generation

was terminated due to the resulted path satisfying the formula $\mathcal{A}\ \mathrm{U}^{[t_l,t_r]}\ \mathcal{G}$. Finally, the sample $\overrightarrow{\mathbf{Q}}$ is updated with the new values in line 18.

Back to Algorithm 12, after the call of function *extendSample* (...) we have a valid sample of observations, using which we can compute the *c. i.* of $Prob\left(s_0,\ \mathcal{A}\ \mathrm{U}^{[t_l,t_r]}\ \mathcal{G}\right)$ as given by Equation (6.31). The latter is done in line 8 and the obtained *c. i.*, if it is tight enough, is checked against the probability constraint $\bowtie\ b$, using Algorithm 3, see lines 10 to 12. If the result is definite then the main cycle terminates and the answer to the model-checking problem is returned in line 14. Note that, as before, we return the error result if we are unable to reach the desired width of the *c. i.*.

To conclude we must say that, in case of $Prob\left(s_0,\ \mathcal{A}\ \mathrm{U}^{[t_l,t_r]}\ \mathcal{G}\right) \neq b$ and no constraints on the sample size, the *a. s.* convergence of the algorithm is guaranteed by the *a. s.* convergence of the *c. i.* borders.

6.5 Conclusion

Recent developments in verification of CSL properties using discrete-event simulation resulted in the model-checking techniques based on hypothesis testing [144, 121, 122]. Unfortunately these techniques support only a subset of CSL, namely the logic without the steady-state operator, and have from three to five main parameters influencing the verification process. In this chapter we devised new and simpler simulation-based algorithms for model-checking all the main operators of CSL: the interval-until, unbounded-until and steady-state operator.

Our algorithms are based on discrete event simulation, sequential confidence intervals and can be easily adapted to model checking of the corresponding PCTL properties. Also, they require only two parameters, that are common for statistical model checking: the desired confidence ξ and the maximum width of the confidence interval δ'. Note that, the algorithms employ a naive sequential procedure for deriving *c. i.* This procedure does not take into account a possible decrease of confidence levels. In the future we plan to employ proper sequential *c. i.* algorithms, such as described in [44, 30].

One of the key assumptions in our approach was that we can deduce the Markov chain structure. The reason for that is that we need to know about the BSCCs of the Markov chain. Since we work with finite state Markov chains, the detection of BSCCs can be done using on-the-fly model generation. Therefore our algorithms can be used without pre-computation of the Markov chain.

To show the feasibility of our approach, in Chapter 7 we provide an experimental comparison between the model-checking techniques given in this chapter, implemented in MRMC v1.3, and the algorithms based on hypothesis testing [122, 144], realized in Ymer v3.0 and VESTA v2.0.

Chapter 7

Experiments

In this chapter we provide a comparative experimental study of the simulation-based model-checking techniques for CSL. We consider the sequential *c. i.* techniques given in Chapter 6 versus the algorithms based on hypothesis testing [122, 144]. The former ones are integrated in MRMC v1.3 (cf. Chapter 2) whereas the latter ones are implemented in Ymer v3.0 and VESTA v2.0 (cf. Section 1.4).

Note that, Ymer and VESTA do not support model checking of the steady-state operator and therefore in the comparison below we only consider the time-interval (time-bounded), and unbounded-until operators. Also we do not discuss model checking of formulas with nested probabilistic operators, e. g., for the property $\mathrm{P}_{\bowtie b}\,(\mathcal{A}\ \mathrm{U}\ \mathcal{G})$ we assume that the sets $\mathcal{A}$ and $\mathcal{G}$ are computed without using simulations. The latter allows us to relax the conditions on the model-checking parameters of VESTA.

Our experiments are aimed at the following main points: *(i)* the verification time, i. e., the required time to verify a formula on a Markov chain; *(ii)* the confidence levels, i. e., the match between the theoretically guaranteed confidence and the one obtained in practice; *(iii)* the peak memory usage (VSZ), i. e., the maximal amount of virtual memory (RAM + swap) needed by the tools during the verification.

For our experiments we have chosen two CTMC case studies: CPS and TQN. The detailed description of these models is given in Section 1.3. These case studies are also used for performance evaluation of various model-checking tools in Section 2.4. As before, care is taken that equivalent input models are used for all the tools.

Further, in Section 7.1 we first introduce the model-checking parameters of Ymer and VESTA, and relate them with the parameters of MRMC. Then, in Section 7.2 the experimental setup is provided. Section 7.3 reports the experimental data and Section 7.4 summarizes the experimental results.

7.1 Tool parameters

For a fair comparison of model-checking techniques it is vital to have the input parameters matching each other in the best possible way. Thus, in this section we present, explain, and relate the main simulation parameters of MRMC, Ymer and VESTA.

MRMC. Recall that the *c. i.*-based techniques for model checking the properties $\mathrm{P}_{\bowtie b}\left(\mathcal{A}\ \mathrm{U}\ \mathcal{G}\right)$ and $\mathrm{P}_{\bowtie b}\left(\mathcal{A}\ \mathrm{U}^{[t_1,t_2]}\ \mathcal{G}\right)$ in a state $s_0 \in S$ (cf. Chapter 6) have two main inputs:

ξ – the desired confidence of the result

δ' – the upper bound on the width of the considered *c. i.*

If for $\widetilde{p} = \mathit{Prob}\left(s_0,\ \mathcal{A}\ \mathrm{U}\ \mathcal{G}\right)$ or $\widetilde{p} = \mathit{Prob}\left(s_0,\ \mathcal{A}\ \mathrm{U}^{[t_1,t_2]}\ \mathcal{G}\right)$ correspondingly we choose δ' such that (cf. Section 108):

$$\delta' \leq |b - \widetilde{p}|, \tag{7.1}$$

then the confidence of getting the correct answer to the model-checking problem is guaranteed to be at least ξ.

Ymer [141]. Sequential hypothesis testing (acceptance sampling) realized in Ymer requires the following main parameters:

α^y – the desired probability of the false-positive answer

β^y – the desired probability of the false-negative answer

δ^y – the half width of the indifference region which is defined as $(b - \delta^y,\ b + \delta^y)$

The false-positive and false-negative probabilities apply if δ^y is chosen in such a way that $\widetilde{p} = \mathit{Prob}\left(s_0,\ \mathcal{A}\ \mathrm{U}^{[t_1,t_2]}\ \mathcal{G}\right)$ does not belong to the indifference region. Recall that Ymer does not support the unbounded-until operator.

Clearly, for a given model-checking problem it is possible to have only one type of incorrect answers, either false positive or false negative. Thus, when matching the parameters of Ymer to those of MRMC, one should take $1 - \xi = \alpha^y = \beta^y$. Also, we should have $\delta' = \delta^y$ because fulfilling Equation (7.1) is equivalent to choosing δ' such that $\widetilde{p}$ does not belong to the open interval $(b - \delta',\ b + \delta')$. The latter is due to considering only the *c. i.* $\left[A_l\left(\overrightarrow{\mathbf{X}}\right),\ A_r\left(\overrightarrow{\mathbf{X}}\right)\right]$ such that $A_r\left(\overrightarrow{\mathbf{X}}\right) - A_l\left(\overrightarrow{\mathbf{X}}\right) < \delta'$. (cf. Section 108).

VESTA [123]. The tool realizes model-checking algorithms based on Monte Carlo simulation of the model and simple hypothesis testing of the samples, as opposed to sequential hypothesis testing realized in Ymer. The model-checking algorithm of VESTA takes the following parameters:

- General:

  α^v – the desired probability of the false-positive answer

  β^v – the desired probability of the false-negative answer

  δ_1^v – the half width of the indifference region

- Specific for unbounded-until formulae:

  $p_\perp^v$ – the stopping probability ($p_\perp^v > 0$)

  δ_2^v – the width of the indifference region for the problem $\mathrm{P}_{=0}\left(\mathcal{A}\ \mathrm{U}\ \mathcal{G}\right)$

The first three parameters are the same as for Ymer and the last two are added specifically for model checking the unbounded-until operator. For the latter, VESTA adds an extra state, called the "termination" state, to the model. Every state of the original model is then extended with a transition to the termination state which is taken with probability $p^v_\perp$ and the existing transition probabilities are renormalized to form proper probability distributions. Such model modification allows VESTA to avoid considering infinite paths when model checking the unbounded-until operator.

To ensure the desired error probabilities α^v and β^v, VESTA requires the following two conditions to be fulfilled:

1. The probability $\widetilde{p} = \mathit{Prob}\,(s_0,\ \mathcal{A}\ \mathrm{U}\ \mathcal{G})$ or $\widetilde{p} = \mathit{Prob}\left(s_0,\ \mathcal{A}\ \mathrm{U}^{[t_1,t_2]}\ \mathcal{G}\right)$ must not lie in the range $(b - \delta^v_1,\ b + \delta^v_1)$.

2. The probability $\widetilde{p} = \mathit{Prob}\,(s_0,\ \mathcal{A}\ \mathrm{U}\ \mathcal{G})$ must not lie in the range:

$$\left(0,\ \frac{\delta^v_2}{p_m^{(|S|-1)} \cdot (1 - p^v_\perp)^{(|S|-1)}}\right] \tag{7.2}$$

   where p_m is the smallest non-zero transition probability in the model.

Note that, since we assume formulas without nested probabilistic operators, the conditions above are the relaxed versions of those given in [123].

Condition 1. is essentially the same as the condition on δ^y for Ymer. Therefore, for the experiments we take $\alpha^v = \beta^v = \alpha^y = \beta^y$ and $\delta^v_1 = \delta^y$.

Condition 2. is new and neither has any direct match in Ymer, because it does not support the unbounded-until operator, nor in MRMC, because our approach does not modify the original model. This condition causes serious problems when model checking large models due to the exponents in the divider of the right border of Interval (7.2). More specifically, the values of p_m and $(1 - p^v_\perp)$ are typically less than one and the state space S consists of millions of states. In this setting the value of $p_m^{(|S|-1)} \cdot (1 - p^v_\perp)^{(|S|-1)}$ becomes extremely small, requiring a drastic decrease of δ^v_2 or $p^v_\perp$ in order to keep the probability $\mathit{Prob}\,(s_0,\ \mathcal{A}\ \mathrm{U}\ \mathcal{G})$ outside the interval (7.2). Moreover, according to [122] the decrease of $p^v_\perp$ dramatically increases the model-checking times[1]. Therefore in our experiments we do not try to satisfy Condition 2 but rather use the default tool values for $p^v_\perp$ and δ^v_2.

7.2 Experimental setup

Every experiment, unless stated otherwise, was repeated 100 times. We report average verification times[2] in milliseconds and use the logarithmic scale for our plots. The memory-usage statistics are collected the same way as described in Section 2.4 and we are interested in the peak virtual-memory usage (VSZ) reported in megabytes. The confidence levels for every tool are computed as the average number of successful model-checking runs for the given experiment.

[1] The same increase of verification time is likely to happen when δ^v_2 is decreased.

[2] We rely on model-checking times reported by the tools.

All experiments were performed on a cluster-computer node with two 2.33 GHz Intel Dual-Core Xeon processors (64-bit) and 16 GB of RAM. The operating system is Suse Linux, because it is supported by all the tools.

The main tool parameters were set as follows: $1 - \xi = \alpha^y = \beta^y = \alpha^v = \beta^v = 0.05$, $\delta' = \delta^y = \delta_1^v = 0.01$, $p_\perp^v = 0.01$ and $\delta_2^v = 0.1$ (cf. Section 7.1).

7.3 Experimental data

Before we proceed with the experimental results, for a better understanding and interpreting the experimental data, let us recall the following important differences between the considered tools:

1. VESTA, unlike MRMC and Ymer, is implemented in Java, therefore:
   (a) VESTA can be expected to be slower when compared to MRMC and Ymer.
   (b) The VSZ values for VESTA reflect the total memory allocated by JVM.
2. VESTA uses simple hypothesis testing whereas Ymer uses sequential. Therefore, we expect VESTA to be slower than Ymer, since (theoretically) to achieve the same level of confidence, sequential hypothesis testing should require a smaller sample size than simple hypothesis testing.
3. MRMC, unlike Ymer and VESTA, works with the pre-generated CTMC, thus:
   (a) The VSZ values for MRMC should reflect the growth of the model size.
   (b) When simulating a path in the model, MRMC has to traverse through the sparse-matrix representation of the CTMC. This can seriously increase the model-checking times on larger models due to a constant need for randomly accessing the elements of a large data structure.
4. Ymer and VESTA, unlike MRMC, do not provide the probability estimates when model checking probabilistic operators. Ymer has a special option that allows to request such estimates. Below we denote the runs of Ymer with this option "on" as Ymer P. Note that Ymer P implements sequential confidence interval based approach described in [108].
5. Ymer and VESTA, unlike MRMC, can only verify properties in the initial state of the model. Thus all further results correspond to model checking CSL formulas in the initial state.
6. As it was discussed in Sections 6.1.3 and 7.1, MRMC requires the *c. i.* of the width that is twice smaller than the width of the corresponding indifference regions of Ymer and VESTA. The latter may cause MRMC to be more accurate because reaching a tighter *c. i.* requires larger statistics. The same reason can cause a lower performance of MRMC due to the large number of required observations.

N	3	6	9	12	15	16	17	18
# states	36	576	6912	73728	737280	1572864	334233	7077888
$Prob\,(true\ \mathrm{U}^{[0,80]}\ busy_1)$	1.0000	0.9999	0.9998	0.9987	0.9951	0.9932	0.9909	0.9882
$Prob\,(true\ \mathrm{U}^{[40,80]}\ serve_1)$	0.9999	0.9988	0.9888	0.9657	0.9326	0.9203	0.9075	0.8944
$Prob\,(poll_1\ \mathrm{U}\ serve_1)$	0.0016	0.0008	0.0005	0.0004	0.0003	0.0003	0.0002	0.0002
$Prob\,(\neg serve_2\ \mathrm{U}\ serve_1)$	0.5213	0.5381	0.5406	0.5405	0.5396	0.5393	0.5389	0.5386

Table 7.1: The numerically-computed probabilities for the CPS case study.

7.3.1 Cyclic Server Polling System (CPS)

For this case study we verified one bounded-until, one interval-until and two unbounded-until formulas on the models with different number of stations (N). The tool settings (cf. Section 7.2) are expected to guarantee the 95% accuracy of the verification results. The exceptions are the time-interval until for all the tools and the unbounded-until formulas when model checked with VESTA. The model state-space sizes and the numerically-computed probabilities for the considered properties are presented in Table 7.1. They help to explain the obtained results.

$\mathrm{P}_{\geq 0.95}\left(true\ \mathrm{U}^{[0,80]}\ busy_1\right)$ – the probability that station 1 becomes busy within 80 time units is at least 0.95. The model-checking times and peak memory consumption for this property are given in Figure 7.1. Note that all the tools showed 100% accuracy.

Figure 7.1(a) indicates that MRMC is (much) faster than Ymer and VESTA. An important observation is that the verification times for Ymer roughly double when the tool is asked to estimate the probabilities. The clear distinction in performance of Ymer and Ymer P is caused by the difference in the used simulation techniques. The latter allowed VESTA to overtake Ymer P after $N = 16$.

Figure 7.1(a) shows the peak memory consumption of the tools. Clearly, Ymer and VESTA both have constant memory usage and the VSZ of MRMC, as predicted, grows when increasing the number of stations N. Remember that both Ymer and VESTA do not generate the complete state space of the input model and the memory needed for sampling seems to be insignificant. Also, the large memory consumption of VESTA is dictated by the amount of memory acquired by the JVM. The behavior depicted in Figure 7.1(b) is the same for all the properties we verified on the CPS case study and therefore we provide it only once.

$\mathrm{P}_{\geq 0.99}\left(true\ \mathrm{U}^{[40,80]}\ serve_1\right)$ – the probability that station 1 is served within the time interval [40, 80] is at least 0.99. Figure 7.2 provides the model-checking times and confidence estimates for this property. As VESTA does not support interval-until formulas, it is not included in the figures.

Figure 7.2(b) shows that the confidence levels for $N \in \{6, 9\}$ are compromised, especially in case of Ymer and Ymer P. This happens because the corresponding probability values (cf. Table 7.1) fall in the indifference region. Moreover, Condition (7.1) required by MRMC for ensuring the confidence $\xi = 0.95$ is also violated. One of the reasons why MRMC provides more accurate answers is that our algorithm first simulates until the *c. i.* is tighter than δ' and then continues simulation until it reaches the

N	2	10	50	100	255	511	1023
# states	15	231	5151	20301	130816	523776	2096128
$Prob\left(true\ \mathrm{U}^{[0,2]}\ full\right)$	0.0262	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
$Prob\left(true\ \mathrm{U}^{[0.5,2]}\ full\right)$	0.0225	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
$Prob\left(true\ \mathrm{U}^{[0,10]}\ full_1\right)$	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
$Prob\left(\neg full_1\ \mathrm{U}\ full_2\right)$	0.0177	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000

Table 7.2: The numerically-computed probabilities for the TQN case study.

definite answer to the model-checking problem. This increases the accuracy because the resulting *c. i.* can be much shorter than δ'. Another reason is that MRMC uses Agresti-Coull *c. i.* that is known to have coverage probability that exceeds the specified confidence levels.

The model-checking times given in Figure 7.2(a) indicate that the accuracy of MRMC comes at a price, see the peak for $N = 9$. In general, MRMC is up to 8 times faster than Ymer P but is slower than Ymer. The performance of the latter one is improving with the growth of N. The reason for that is likely to be the rapid increase of distance between the values of $Prob\left(true\ \mathrm{U}^{[40,80]}\ serve_1\right)$, cf. Table 7.1, and the probability bound of the formula.

$\mathrm{P}_{\geq 0.2}\left(poll_1\ \mathrm{U}\ serve_1\right)$ – the probability that station 1 is served after being polled is at least 0.2. Both MRMC and VESTA showed 100% accuracy when model checking this property. The performance results given in Figure 7.3(a) indicate that the time required by VESTA is almost constant for all model sizes. In general MRMC is at least 10 times faster than VESTA.

$\mathrm{P}_{\geq 0.5}\left(\neg serve_2\ \mathrm{U}\ serve_1\right)$ – the probability that station 1 is served before station 2 is at least 0.5. Figure 7.3(b) provides the model-checking times for MRMC which once again showed 100% accuracy in model-checking results. The plots for VESTA are not present because it did not terminate within the 15 minutes time-out (compared to seconds required by MRMC).

7.3.2 Tandem Queuing Network (TQN)

For this case study we verified two bounded-until, one interval-until and one unbounded-until operator on the models with different queue capacities (N). The tool settings (cf. Section 7.2) are expected to guarantee the 95% accuracy of the verification results. The numerically-computed probabilities for the considered properties, cf. Table 7.2, help to explain the obtained results. Note that, since the parameter N is changed in a non-linear manner, the horizontal axis of the plots given in this section is logarithmic.

$\mathrm{P}_{\leq 0.01}\left(true\ \mathrm{U}^{[0,2]}\ full\right)$ – the probability that both queues become full within 2 time units is at most 0.01. The peak memory consumption and confidence estimates for this property are given in Figure 7.4.

The peak memory usage in Figure 7.4(a) is similar to that of the CPS case study. As before, the memory consumption is the same for all the properties we verified and therefore we provide it only once.

The confidence estimates in Figure 7.4(b) exhibit a slight decrease of confidence for Ymer and VESTA at $N = 2$. This can be explained by the fact that the corresponding value of $Prob\left(true\ \mathrm{U}^{[0,2]}\ full\right)$ (cf. Table 7.2) is relatively close to the probability bound. Still, the confidence levels stay above the theoretically-predicted level.

The model-checking times are given in Figure 7.5(a). There are no results for Ymer P because it was not terminating within the 15 minutes time-out. Similarly to the previous results, MRMC is generally faster than the other tools. The peak in verification times at $N = 2$ is the price MRMC pays for being 100% accurate.

$\mathrm{P}_{\leq 0.1}\left(true\ \mathrm{U}^{[0.5,2]}\ full\right)$ – the probability that both queues become full within time interval [0.5, 2] is at most 0.1. For this property all the tools showed 100% accuracy. Once again, Ymer P was not able to finish verification within 15 minutes and therefore is not present. The performance results given in Figure 7.5(b) show the behavior similar to the one for $\mathrm{P}_{\leq 0.01}\left(true\ \mathrm{U}^{[0,2]}\ full\right)$.

$\mathrm{P}_{\leq 0.98}\left(true\ \mathrm{U}^{[0,10]}\ full_1\right)$ – the probability that the first queue becomes full within 10 time units is at most 0.98. In this case Ymer P coped with the property fine and all the tools were 100% accurate. The model-checking times, given in Figure 7.6(a), are similar to the ones for the previous two properties.

$\mathrm{P}_{\leq 0.03}\left(\neg full_1\ \mathrm{U}\ full_2\right)$ – the probability that the second queue becomes full before the first queue is at most 0.03. Both, VESTA and MRMC were completely accurate in their model-checking results. The verification times shown in Figure 7.6(b) reflect that MRMC is at least 6 times faster than VESTA which exhibits asymptotic model-checking times. The times for MRMC, as it is the case for all considered properties, steadily grow with the size of the model.

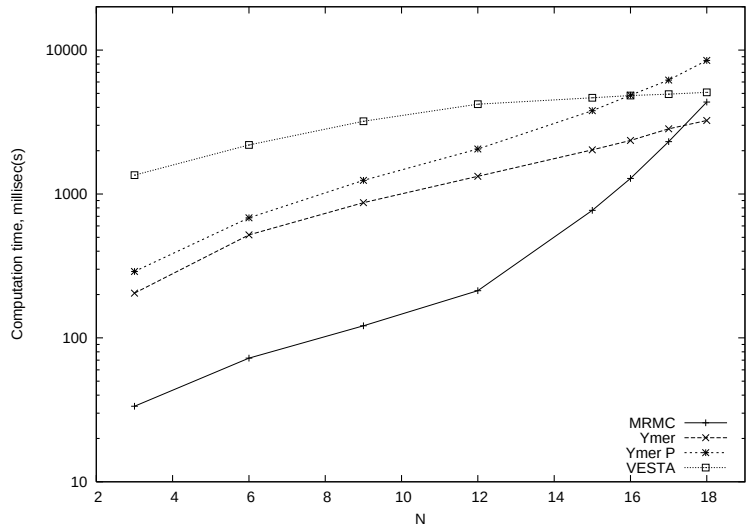
7.4 Conclusion

In this chapter we presented the experimental comparison between the model-checking techniques given in Chapter 6, implemented in MRMC v1.3, and the algorithms based on hypothesis testing [122, 144], realized in Ymer v3.0 and VESTA v2.0. The tool parameters and the input models were matched as closely as possible, see Section 7.1, providing a relatively fair comparison obscured only by the tool differences explained in the beginning of Section 7.3. Each experiment was repeated 100 times guaranteeing a good confidence in the results of this comparison.

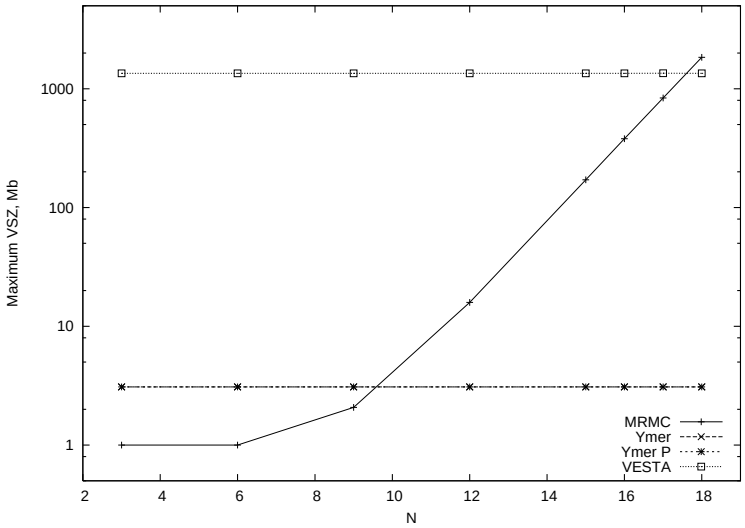
It was shown that all the tools fulfill the theoretically-predicted confidence levels. Moreover, very often the provided accuracy was reaching 100% which can be explained by the relatively large distance between the estimated probabilities and the probability bounds of the properties. Unlike for Ymer and VESTA, the peak-memory consumption of MRMC grows linearly with the growth of the model sizes. The latter is due to using the pre-generated Markov chain, as opposed to the on-demand state-space generation.

On average, and especially on smaller models, MRMC was several times faster than the other tools. Also, for some properties Ymer P and VESTA were not able to provide answers within a reasonable time. Note that Ymer P uses sequential confidence interval based approach described in [108]. Another important observation is that the verification times of MRMC grow faster, with the increase of model sizes, than the times of Ymer and VESTA. We anticipate that this is due to the use of the pre-generated state space but further investigation is required to obtain more insight into this phenomenon.

The results obtained on TQN and CPS case studies show that in most cases MRMC runs faster, provides more accurate results, and can handle more properties than the above mentioned tools. The reasons for that might be: a more efficient implementation; the use of Agresti-Coull *c. i.*; specific sequential procedure; and the use of the *c. i.* that are tighter than the indifference regions used in tools based on hypothesis testing. Our results do not necessarily mean that the algorithms suggested in Chapter 6 are more efficient than the ones realized in Ymer and VESTA. For a better comparison of the underlying techniques, we need to compare the required sample sizes, since the latter is a common criteria for evaluating the efficiency of simulation based algorithms.

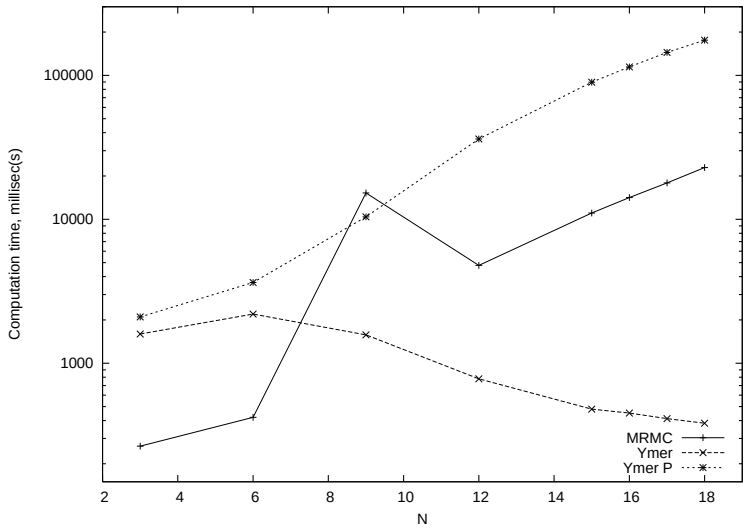


(a) model check time

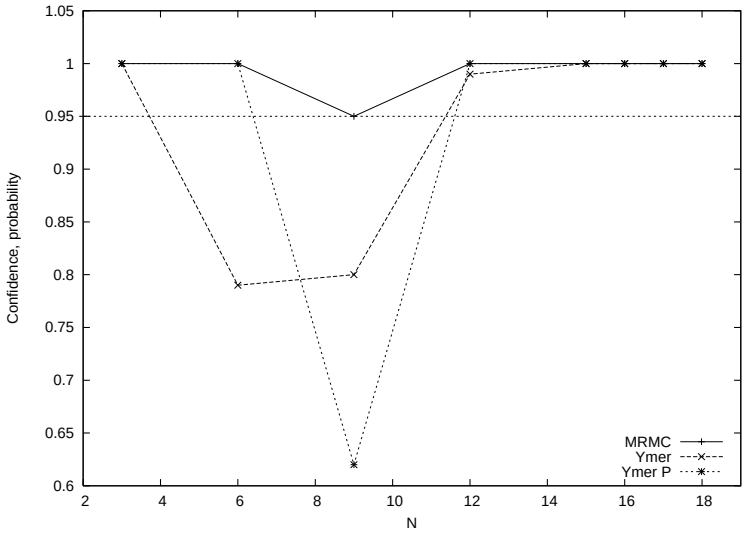


(b) peak memory

Figure 7.1: CPS : $\mathrm{P}_{\geq 0.95}\left(\mathit{true}\ \mathrm{U}^{[0,80]}\ \mathit{busy}_1\right)$

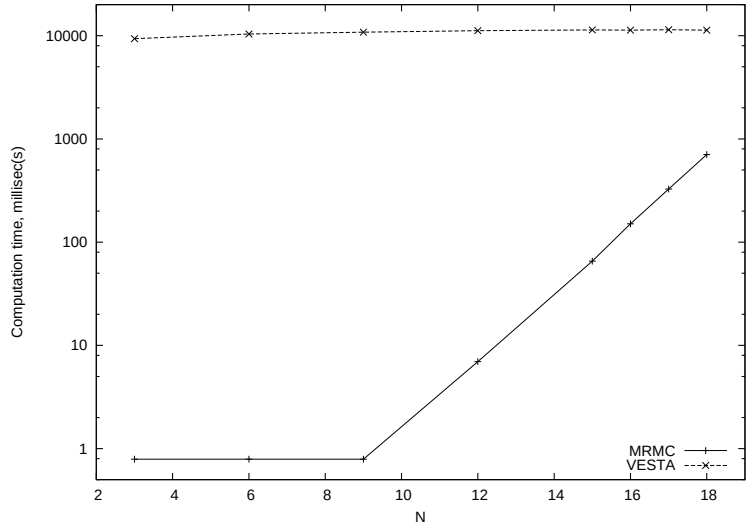


(a) model check time

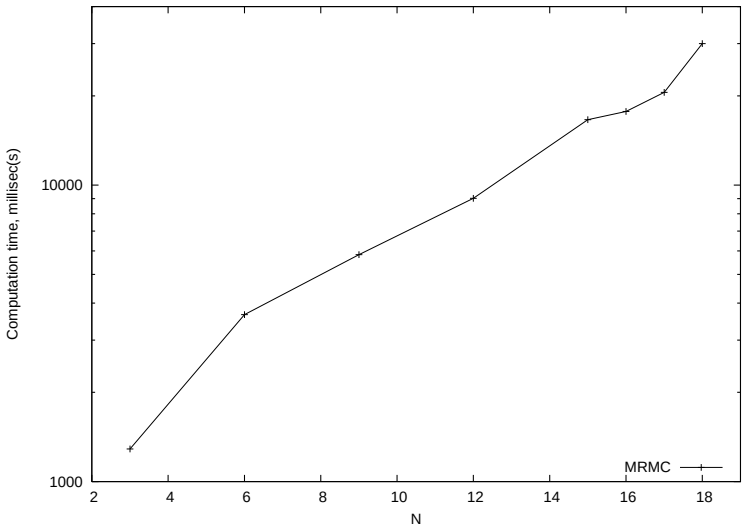


(b) confidence

Figure 7.2: CPS : $\mathrm{P}_{\geq 0.99}\left(\mathit{true}\ \mathrm{U}^{[40,80]}\ \mathit{serve}_1\right)$

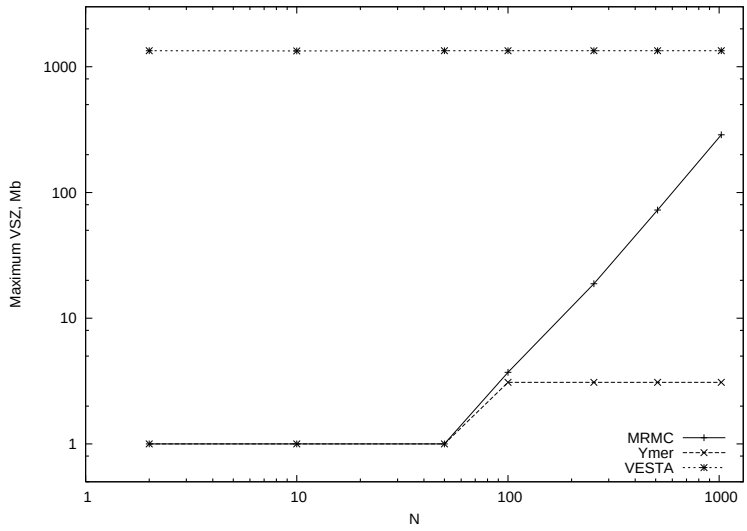


(a) $\mathrm{P}_{\geq 0.2}\,(poll_1\ \mathrm{U}\ serve_1)$

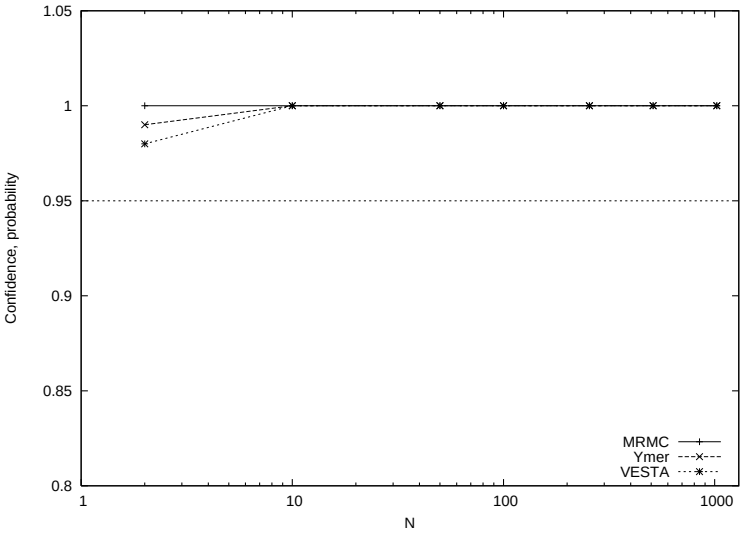


(b) $\mathrm{P}_{\geq 0.5}\,(\neg serve_2\ \mathrm{U}\ serve_1)$

Figure 7.3: CPS, model check time

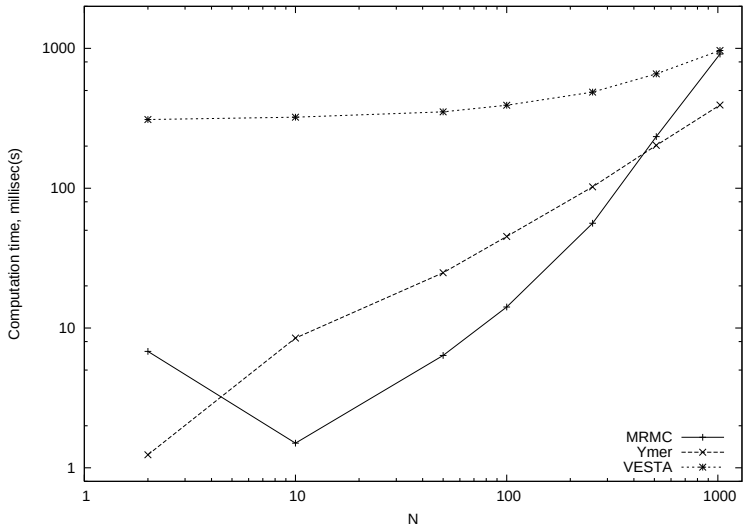


(a) peak memory

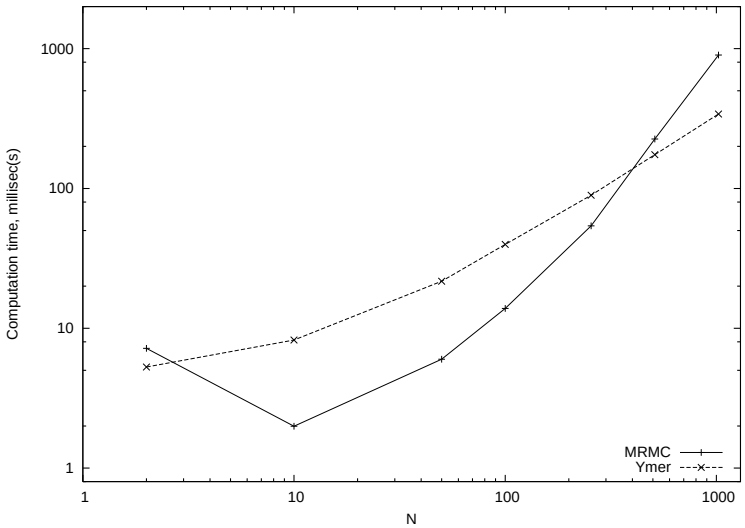


(b) confidence

Figure 7.4: TQN : $\mathrm{P}_{\leq 0.01}\left(\mathit{true}\ \mathrm{U}^{[0,2]}\ \mathit{full}\right)$

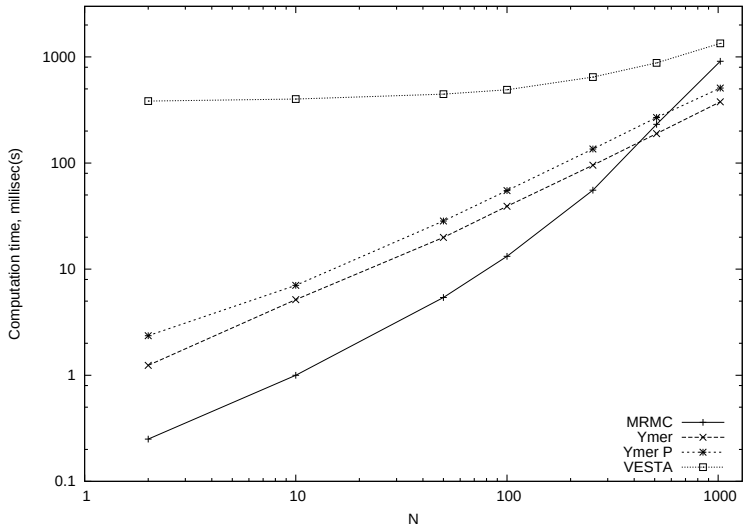


(a) $\mathrm{P}_{\leq 0.01}\left(\mathit{true}\ \mathrm{U}^{[0,2]}\ \mathit{full}\right)$

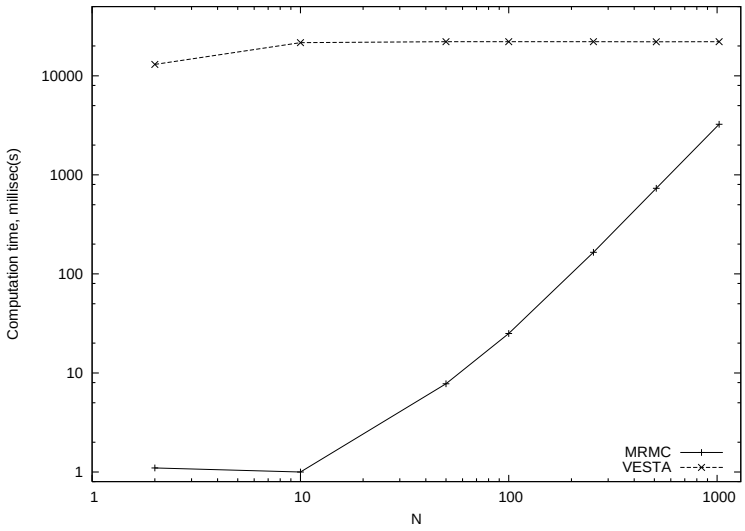


(b) $\mathrm{P}_{\leq 0.1}\left(\mathit{true}\ \mathrm{U}^{[0.5,2]}\ \mathit{full}\right)$

Figure 7.5: TQN, model check time



(a) $\mathrm{P}_{\leq 0.98}\left(\mathit{true}\ \mathrm{U}^{[0,10]}\ \mathit{full}_1\right)$



(b) $\mathrm{P}_{\leq 0.03}\left(\neg \mathit{full}_1\ \mathrm{U}\ \mathit{full}_2\right)$

Figure 7.6: TQN, model check time

Part III

Conclusion

Chapter 8

Concluding remarks

In this work, we contributed to four important aspects of probabilistic model checking:

- *The development of an efficient model checker:* our tool, called MRMC, supports model checking of DTMCs, CTMCs and MRMs.
- *The improvement of model-checking algorithms:* we developed a precise on-the-fly steady-state detection procedure for time-bounded until operator of CSL.
- *The efficiency of the state-space reduction techniques:* we empirically investigated the effect of bisimulation minimization.
- *The development of simulation-based model-checking techniques:* we proposed effective verification algorithms for all main operators of CSL.

Below we briefly summarize our results and provide future research directions. Note that, more detailed conclusions are provided at the end of the corresponding chapters.

Markov Reward Model Checker. We developed a model checker that supports DTMC, CTMC and MRM models, allowing for model checking PRCTL and CSRL properties. Our experiments show that MRMC is highly competitive with other tools, especially when applied to models that have up to several million states. The tool is used in several third-party projects aimed at: the validation and performance evaluation of Stochastic Well-formed Nets, counter-examples generation, and model checking CTMDPs.

In the future, we expect to extend MRMC with the algorithms for CTMDP model checking [12], counter-example generation [55], simulation-based model checking algorithms and state-space reduction techniques, such as backwards and weak bisimulation [13].

Steady-State Detection for Time-Bounded Reachability. In this study first we refined error bounds for existing standard transient analysis and for time-bounded reachability algorithms that incorporate on-the-fly steady-state detection and we devised a simple technique for precise steady-state detection. Our backward algorithm increases the runtime by a factor two, and requires two extra probability vectors. For

the forward algorithm there is no increase of run time, and no additional space is required. In both cases the approach guarantees the avoidance of premature steady-state detection. Note that, although for the backward algorithm the computation time is doubled (prior to reaching the steady-state, at some time t') the verification time for after approximately $2 \cdot t'$ is reduced.

One of the possible extensions of this work is to derive a simple technique for estimating the convergence rate of CTMCs. This would allow to avoid using precise steady-state detection when the steady-state is sure not to be reached, thus optimizing the performance.

Bisimulation Minimization. Our experiments show that, like in traditional model checking, in probabilistic setting using bisimulation minimization can result in up to exponential state-space reduction. On top of that, and unlike in traditional model checking, a substantial reduction in verification time can be obtained (up to a factor 50). For measure-driven bisimulation for models without rewards, this speedup comes with almost no memory penalty whereas for ordinary bisimulation some experiments showed an increase of peak memory up to 50%. In our case studies with rewards, we experienced a reduction in peak memory use. In general, we observed that verification-time reductions strongly depend on the number of transitions in the Markov chain, its structure, and the convergence rate of numerical computations. A case study using bisimulation minimization and symmetry reduction showed that, although the latter one is faster, the former one provides a much coarser state-space partitioning.

In the future we plan to investigate combinations of symmetry reduction with bisimulation minimization, and to extend our experimental work towards MDPs and simulation preorders.

Model Checking by Discrete Event Simulation. Up till now, the simulation algorithms for verifying CSL properties on CTMCs did not support the steady-state operator. Moreover, these algorithms, based on hypothesis testing, have too many parameters, influencing the verification process, and the latter complicates their usability. In our work we devised new and simple algorithms, based on sequential confidence intervals, that allow for model checking of all the main CSL operators: the interval until, unbounded until and steady state. These algorithms can be easily adapted to model checking of the corresponding PCTL properties and also require only two parameters that are common to statistical model checking: the desired confidence and the maximum width of the confidence interval. The experiments show that the MRMC based implementation of the suggested algorithms is generally faster and provides more accurate results than the model checkers based on hypothesis testing (Ymer, VESTA). In addition, MRMC could handle more properties.

Unfortunately, our experimental comparison does not take into account some key differences between the considered tools and the underlying algorithms. Therefore, we plan to perform further investigation on the experimental as well as algorithmic levels. For that we need to study and compare the sample sizes required by the considered techniques and to perform more experiments.

Bibliography

[1] Gul Agha, José Meseguer, and Koushik Sen. PMaude: Rewrite-based Specification Language for Probabilistic Object Systems. *Electronic Notes in Theoretical Computer Science*, 153(2):213–239, 2006.

[2] Husain Aljazzar and Stefan Leue. Extended Directed Search for Probabilistic Timed Reachability. In Eugene Asarin and Patricia Bouyer, editors, *Formal Modeling and Analysis of Timed Systems (FORMATS)*, volume 4202 of *LNCS*, pages 33–51. Springer, 2006.

[3] Rajeev Alur and Thomas A. Henzinger. Reactive modules. *Formal Methods in System Design*, 15(1):7–48, 1999.

[4] Suzana Andova, H. Hermanns, and Joost-Pieter Katoen. Discrete-Time Rewards Model-Checked. In K.G. Larsen and P. Niebert, editors, *Formal Modeling and Analysis of Timed Systems (FORMATS)*, volume 2791, pages 88–104. LNCS, Springer, 2003.

[5] Adnan Aziz, Kumud Sanwal, Vigyan Singhal, and Robert Brayton. Model-checking continuous-time Markov chains. *ACM Transactions on Computational Logic*, 1(1):162–170, 2000.

[6] Adnan Aziz, Kumud Sanwal, Vigyan Singhal, Robert K. Brayton, and Alberto L. Sangiovanni-Vincentelli. It Usually Works: The Temporal Logic of Stochastic Systems. In P. Wolper, editor, *Computer Aided Verification (CAV)*, volume 939 of *LNCS*, pages 155–165. Springer, 1995.

[7] R. Iris Bahar, Erica A. Frohm, Charles M. Gaona, Gary D. Hachtel, Enrico Macii, Abelardo Pardo, and Fabio Somenzi. Algebraic decision diagrams and their applications. In *International Conference on Computer-Aided Design (ICCAD)*, pages 188–191. IEEE Computer Society, 1993.

[8] C. Baier, B. Haverkort, H. Hermanns, and J.-P. Katoen. Model-Checking Algorithms for Continuous-Time Markov Chains. *IEEE Transactions on Software Engineering*, 29(6):524–541, 2003.

[9] Christel Baier, Frank Ciesinski, and Marcus Größer. ProbMela and verification of Markov decision processes. *ACM SIGMETRICS Performance Evaluation Review*, 32(4):22–27, 2005.

[10] Christel Baier, Boudewijn R. Haverkort, Holger Hermanns, and Joost-Pieter Katoen. Model Checking Continuous-Time Markov Chains by Transient Analysis. In E. Allen Emerson and A. Prasad Sistla, editors, *Computer Aided Verification (CAV)*, volume 1855 of *LNCS*, pages 358–372. Springer, 2000.

[11] Christel Baier, Boudewijn R. Haverkort, Holger Hermanns, and Joost-Pieter Katoen. On the Logical Characterisation of Performability Properties. In Ugo Montanari, Jos D. P. Rolim, and Emo Welzl, editors, *International Colloquium on Automata, Languages and Programming (ICALP)*, volume 1853 of *LNCS*, pages 780–792. Springer, 2000.

[12] Christel Baier, Holger Hermanns, Joost-Pieter Katoen, and Boudewijn R. Haverkort. Efficient computation of time-bounded reachability probabilities in uniform continuous-time Markov decision processes. *Theoretical Computer Science*, 345(1):2–26, 2005.

[13] Christel Baier, Holger Hermanns, Joost-Pieter Katoen, and Verena Wolf. Bisimulation and Simulation Relations for Markov Chains. *Electronic Notes in Theoretical Computer Science*, 162:73–78, 2006.

[14] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. MIT Press, Cambridge, MA, USA, 2008.

[15] Christel Baier, Joost-Pieter Katoen, Holger Hermanns, and Verena Wolf. Comparative branching-time semantics for Markov chains. *Information and Computation*, 200(2):149–214, 2005.

[16] Alexander Bell. *Distributed evaluation of stochastic Petri nets*. PhD thesis, Rheinisch-Westfalische Technischen Hochschule Aachen, Aachen, Germany, 2004.

[17] S. Bernardi, S. Donatelli, and A. Horváth. Compositionality in the GreatSPN Tool and Its Application to the Modelling of Industrial Applications. In K. Jensen, editor, *Practical Use of High-level Petri Nets*, pages 127–146. University of Aarhus, Department of Computer Science, 2000.

[18] P. Billingsley. *Probability and Measure*. John Wiley & Sons, New York, NY, USA, 3d edition, 1995.

[19] Eckard Bode, Marc Herbstritt, Holger Hermanns, Sven Johr, Thomas Peikenkamp, Reza Pulungan, Ralf Wimmer, and Bernd Becker. Compositional Performability Evaluation for STATEMATE. In *Quantitative Evaluation of Systems (QEST)*, pages 167–178. IEEE Computer Society, 2006.

[20] Barry W. Boehm, Clark, Horowitz, Brown, Reifer, Chulani, Ray Madachy, and Bert Steece. *Software Cost Estimation with Cocomo II*. Prentice Hall PTR, 2000.

[21] A. Bondavalli, A. Coccoli, and F. Di Giandomenico. *QoS Analysis of Group Communication Protocols in Wireless Environment*. Kluwer Academic Publishers Concurrency in Dependable Computing, 2002.

[22] Lawrence D. Brown, T. Tony Cai, and Anirban DasGupta. Interval estimation for a binomial proportion. *Statistical Science*, 16(2):101–133, 2001.

[23] A. Bryant and J. A. Kirkham. B. W. Boehm software engineering economics: a review essay. *SIGSOFT Software Engineering Notes*, 8(3):44–60, 1983.

[24] P. Buchholz. Exact and ordinary lumpability in finite Markov chains. *Journal of Applied Probability*, 31:59–75, 1994.

[25] P. Buchholz, M. Fischer, P. Kemper, and C. Tepper. Model checking of CTMCs and discrete event simulation integrated in the APNN-Toolbox. In F. Bause, editor, *Measurement, Modelling, and Evaluation of Computer-Communication Systems*, volume 781, pages 30–33. Fachbereich Informatik, Universität Dortmund, 2003.

[26] P. Buchholz, J.-P. Katoen, P. Kemper, and C. Tepper. Model-checking large structured Markov chains. *Journal of Logic and Algebraic Programming*, 56:69–96, 2003.

[27] Christos G. Cassandras and Stephane Lafortune. *Introduction to Discrete Event Systems*. Kluwer Academic Publishers, Dordrecht, The Netherlands, 1999.

[28] D. Cerotti, D. D'Aprile, S. Donatelli, and J. Sproston. Verifying Stochastic Well-formed Nets with CSL Model-Checking Tools. In K. Goossens and L. Petrucci, editors, *Application of Concurrency to System Design (ACSD)*, pages 143–152. IEEE Computer Society, 2006.

[29] Byron Changuion, Ian Davies, and Micheal Nelte. DaNAMiCS: a Petri Net Editor. http://www.cs.uct.ac.za/Research/DNA/microweb/danamics/DNAFrameH.html, 2007.

[30] Y. S. Chow and H. Robbins. On the asymptotic theory of fixed-width sequential confidence intervals for the mean. *Annals of Mathematical Statistics*, 36(2):456–462, 1965.

[31] Frank Ciesinski and Marcus Größer. On Probabilistic Computation Tree Logic. In Christel Baier, Boudewijn R. Haverkort, Holger Hermanns, Joost-Pieter Katoen, and Markus Siegle, editors, *Validation of Stochastic Systems*, volume 2925 of *LNCS*, pages 147–188. Springer, 2004.

[32] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *AMC Transactions On Programming Languages And Systems*, 8(2):244–263, 1986.

[33] L. Cloth, J.-P. Katoen, M. Khattri, and R. Pulungan. Model-Checking Markov Reward Models with Impulse Rewards. In *Dependable Systems and Networks (DSN)*, pages 722–731. IEEE Computer Society, 2005.

[34] Lucia Cloth. *Model Checking Algorithms for Markov Reward Models*. PhD thesis, University of Twente, Enschede, The Netherlands, 2006.

[35] The GNU Compiler Collection. GCC web page. http://gcc.gnu.org/, 2008.

[36] David R. Cox. A use of complex probabilities in the theory of stochastic processes. In *Cambridge Philosophical Society*, volume 51, pages 313–319, 1955.

[37] M. A. Crane and D. L. Iglehart. Simulating Stable Stochastic Systems III: Regenerative Processes and Discrete-Event Simulations. *Operations Research*, 23:33–45, 1975.

[38] Davide D'Aprile, Susanna Donatelli, and Jeremy Sproston. CSL Model Checking for the GreatSPN Tool. In Cevdet Aykanat, Tugrul Dayar, and Ibrahim Korpeoglu, editors, *Computer and Information Sciences*, volume 3280 of *LNCS*, pages 543–553. Springer, 2004.

[39] Pedro R. D'Argenio, Bertrand Jeannet, Henrik Ejersbo Jensen, and Kim Guldstrand Larsen. Reachability Analysis of Probabilistic Systems by Successive Refinements. In Luca de Alfaro and Stephen Gilmore, editors, *Process Algebra and Probabilistic Methods, Performance Modeling and Verification (PAPM/PROBMIV)*, volume 2165 of *LNCS*, pages 39–56. Springer, 2001.

[40] S. Derisavi. *Solution of Large Markov Models Using Lumping Techniques and Symbolic Data Structures*. PhD thesis, University of Illinois at Urbana-Champaign, 2005.

[41] Salem Derisavi, Holger Hermanns, and William H. Sanders. Optimal State-Space Lumping in Markov Chains. *Information Processing Letters*, 87(6):309–315, 2003.

[42] R. M. Dudley. *Real Analysis and Probability*. Cambridge Studies in Advanced Mathematics. Cambridge University Press, Cambridge, UK, 2002.

[43] Harald Fecher, Martin Leucker, and Verena Wolf. *Don't Know* in Probabilistic Systems. In Antti Valmari, editor, *Model Checking of Software (SPIN)*, volume 3925 of *LNCS*, pages 71–88. Springer, 2006.

[44] George S. Fishman. *Monte Carlo: Concepts, Algorithms and Applications*. Springer, New York, NY, USA, 1996.

[45] K. Fisler and M. Y. Vardi. Bisimulation Minimization in an Automata-Theoretic Verification Framework. In Ganesh Gopalakrishnan and Phillip J. Windley, editors, *Formal Methods in Computer-Aided Design (FMCAD)*, volume 1522 of *LNCS*, pages 115–132. Springer, 1998.

[46] K. Fisler and M. Y. Vardi. Bisimulation and Model Checking. In Laurence Pierre and Thomas Kropf, editors, *Correct Hardware Design and Verification Methods (CHARME)*, volume 1703 of *LNCS*, pages 338–342. Springer, 1999.

[47] K. Fisler and M. Y. Vardi. Bisimulation Minimization and Symbolic Model Checking. In *Formal Methods in System Design*, volume 21, pages 39–78. Kluwer Academic Publishers, 2002.

[48] W. Fokkink and J. Pang. Simplifying Itai-Rodeh leader election for anonymous rings. *Electronic Notes in Theoretical Computer Science*, 128(6):53–68, 2004.

[49] Eclipse Foundation. Eclipse web page. http://www.eclipse.org, 2007.

[50] Bennett L. Fox and Peter W. Glynn. Computing Poisson probabilities. *Communications of the ACM*, 31(4):440–445, 1988.

[51] M. Fujita, P. C. McGeer, and J. C.-Y. Yang. Multi-Terminal Binary Decision Diagrams: An Efficient Data Structure for Matrix Representation. *Formal Methods in System Design*, 10(2-3):149–169, 1997.

[52] E. Gafni and M. Mitzenmacher. Analysis of Timing-Based Mutual Exclusion with Random Times. In *Principles of Distributed Computing*, pages 13–21, 1999.

[53] Marcus Größer and Christel Baier. Partial Order Reduction for Markov Decision Processes: A Survey. In Frank S. de Boer, Marcello M. Bonsangue, Susanne Graf, and Willem P. de Roever, editors, *Formal Methods for Components and Objects (FMCO)*, volume 4111 of *LNCS*, pages 408–427. Springer, 2005.

[54] Rajiv Gupta, Scott A. Smolka, and Shaji Bhaskar. On randomization in sequential and distributed algorithms. *ACM Computing Surveys*, 26(1):7–86, 1994.

[55] Tingting Han and Joost-Pieter Katoen. Counterexamples in Probabilistic Model Checking. In Orna Grumberg and Michael Huth, editors, *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 4424 of *LNCS*, pages 72–86. Springer, 2007.

[56] N. Hansson and B. Jonsson. A logic for reasoning about time and reliability. *Formal Aspects of Computing*, 6(5):512–535, 1994.

[57] B. Haverkort, L. Cloth, H. Hermanns, J.-P. Katoen, and C. Baier. Model Checking Performability Properties. In *Dependable Systems and Networks (DSN)*, pages 103–112. IEEE Computer Society, 2002.

[58] B. Haverkort, H. Hermanns, and J.-P. Katoen. On the Use of Model Checking Techniques for Dependability Evaluation. In *Symposium on Reliable Distributed Systems (SRDS)*, pages 228–237. IEEE Computer Society, 2000.

[59] Boudewijn R. Haverkort. *Performance of Computer Communication Systems: A Model-Based Approach*. John Wiley & Sons, Inc., New York, NY, USA, 1998.

[60] H. Hermanns, J.-P. Katoen, J. Meyer-Kayser, and M. Siegle. A Tool for Model Checking Markov Chains. *Int. J. on Softw. for Technology Transfer (STTT)*, 4(2):153–172, 2003.

[61] Holger Hermanns, Ulrich Herzog, Ulrich Klehmet, Vassilis Mertsiotakis, and Markus Siegle. Compositional performance modelling with the TIPPtool. *Performance Evaluation*, 39(1-4):5–35, 2000.

[62] Holger Hermanns and Sven Johr. Uniformity by Construction in the Analysis of Nondeterministic Stochastic Systems. In *Dependable Systems and Networks (DSN)*, pages 718–728. IEEE Computer Society, 2007.

[63] Holger Hermanns, Joost-Pieter Katoen, Joachim Meyer-Kayser, and Markus Siegle. A Markov Chain Model Checker. In Susanne Graf and Michael Schwartzbach, editors, *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 1785 of *LNCS*, pages 347–362. Springer, 2000.

[64] Holger Hermanns, Joost-Pieter Katoen, Joachim Meyer-Kayser, and Markus Siegle. Towards Model Checking Stochastic Process Algebra. In Wolfgang Grieskamp, Thomas Santen, and Bill Stoddart, editors, *Integrated Formal Methods (IFM)*, volume 1945 of *LNCS*, pages 420–439. Springer, 2000.

[65] Holger Hermanns, Marta Z. Kwiatkowska, Gethin Norman, David Parker, and Markus Siegle. On the use of MTBDDs for performability analysis and verification of stochastic systems. *Journal of Logic and Algebraic Programming*, 56(1-2):23–67, 2003.

[66] Holger Hermanns, Joachim Meyer-Kayser, and Markus Siegle. Multi Terminal Binary Decision Diagrams to Represent and Analyse Continuous Time Markov Chains. In B. Plateau, W. J. Stewart, and M. Silva, editors, *Numerical Solutions of Markov Chains*, pages 188–207. Prensas Universitarias, 1999.

[67] Jane Hillston. *A Compositional Approach to Performance Modelling*. Distinguished Dissertations Series. Cambridge University Press, New York, NY, USA, 1996.

[68] A. Hinton, M. Kwiatkowska, G. Norman, and D. Parker. PRISM: A Tool for Automatic Verification of Probabilistic Systems. In H. Hermanns and J. Palsberg, editors, *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 3920 of *LNCS*, pages 441–444. Springer, 2006.

[69] Robert V. Hogg and Allen T. Craig. *Introduction to Mathematical Statistics*. MacMillan, New York, NY, USA, 4th edition, 1978.

[70] A. Hordijk, D. L. Iglehart, and R. A. Schassberger. Discrete Time Methods for Simulating Continuous Time Markov Chains. *Advances in Applied Probability*, 8:772–788, 1976.

[71] Michael Huth. An Abstraction Framework for Mixed Non-deterministic and Probabilistic Systems. In Christel Baier, Boudewijn R. Haverkort, Holger Hermanns, Joost-Pieter Katoen, and Markus Siegle, editors, *Validation of Stochastic Systems*, volume 2925 of *LNCS*, pages 419–444. Springer, 2004.

[72] Michael Huth. On finite-state approximants for probabilistic computation tree logic. *Theoretical Computer Science*, 346(1):113–134, 2005.

[73] Oliver C. Ibe and Kishor S. Trivedi. Stochastic Petri Net Models of Polling Systems. *Selected Areas in Communications*, 8(9):1649–1657, 1990.

[74] Free Software Foundation Inc. Yacc and Lex web page. http://dinosaur.compilertools.net/, 2008.

[75] Scientific Toolworks Inc. Understand C/C++. http://www.scitools.com/, 2008.

[76] Alon Itai and Michael Rodeh. Symmetry breaking in distributed networks. *Information and Computation*, 88(1):60–87, 1990.

[77] Lloyd R. Jaisingh. *Statistics for the Utterly Confused*. McGraw-Hill, New York, NY, USA, second edition, 2005.

[78] David N. Jansen, Joost-Pieter Katoen, Marcel Oldenkamp, Mariëlle Stoelinga, and Ivan S. Zapreev. How Fast and Fat Is Your Probabilistic Model Checker? In *Haifa Verification Conference (HVC)*, volume 4899 of *LNCS*, pages 65 – 79. Springer, 2008.

[79] Sven Johr. *Model Checking Compositional Markov Systems*. PhD thesis, Universität des Saarlandes, Saarbrücken, Germany, 2007.

[80] Samuel Karlin and James L. McGregor. The differential equations of birth-and-death processes, and the Stieltjes moment problem. *Transactions of the American Mathematical Society*, 85(2):489–546, 1957.

[81] J.-P. Katoen, M. Kwiatkowska, G. Norman, and D. Parker. Faster and Symbolic CTMC Model Checking. In Luca de Alfaro and Stephen Gilmore, editors, *Process Algebra and Probabilistic Methods, Performance Modeling and Verification (PAPM/PROBMIV)*, volume 2165 of *LNCS*, pages 23–38. Springer, 2001.

[82] J.-P. Katoen and Ivan S. Zapreev. Safe On-The-Fly Steady-State Detection for Time-Bounded Reachability. Technical Report TR-CTIT-05-52, CTIT, University of Twente, 2005.

[83] Joost-Pieter Katoen, Tim Kemna, Ivan S. Zapreev, and David N. Jansen. Bisimulation Minimisation Mostly Speeds Up Probabilistic Model Checking. In Orna Grumberg and Michael Huth, editors, *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 4424 of *LNCS*, pages 87–101. Springer, 2007.

[84] Joost-Pieter Katoen, Maneesh Khattri, and Ivan S. Zapreev. A Markov Reward Model Checker. In *Quantitative Evaluation of Systems (QEST)*, pages 243–244. IEEE Computer Society, 2005.

[85] Joost-Pieter Katoen, Daniel Klink, Martin Leucker, and Verena Wolf. Three-Valued Abstraction for Continuous-Time Markov Chains. In Werner Damm and Holger Hermanns, editors, *Computer Aided Verification (CAV)*, volume 4590 of *LNCS*, pages 311–324. Springer, 2007.

[86] Joost-Pieter Katoen and Ivan S. Zapreev. Safe On-The-Fly Steady-State Detection for Time-Bounded Reachability. In *Quantitative Evaluation of Systems (QEST)*, pages 301–310. IEEE Computer Society, 2006.

[87] M. Kwiatkowska. Model Checking for Probability and Time: From Theory to Practice. In *Logic in Computer Science (LICS)*, pages 351–360. IEEE Computer Society, 2003.

[88] M. Kwiatkowska, G. Norman, and D. Parker. PRISM: Probabilistic Symbolic Model Checker. In T. Field, P. Harrison, J. Bradley, and U. Harder, editors, *Modelling Techniques and Tools for Computer Performance Evaluation (TOOLS)*, volume 2324 of *LNCS*, pages 200–204. Springer, 2002.

[89] M. Kwiatkowska, G. Norman, and D. Parker. Probabilistic Symbolic Model Checking with PRISM: A Hybrid Approach. *International Journal on Software Tools for Technology Transfer (STTT)*, 6(2):128–142, 2004.

[90] M. Kwiatkowska, G. Norman, and D. Parker. Symmetry Reduction for Probabilistic Model Checking. In T. Ball and R. Jones, editors, *Computer Aided Verification (CAV)*, volume 4114 of *LNCS*, pages 234–248. Springer, 2006.

[91] M. Kwiatkowska, D. Parker, Y. Zhang, and R. Mehmood. Dual-Processor Parallelisation of Symbolic Probabilistic Model Checking. In D. DeGroot and P. Harrison, editors, *Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pages 123–130. IEEE Computer Society, 2004.

[92] Marta Kwiatkowska, Gethin Norman, and David Parker. Game-based Abstraction for Markov Decision Processes. In *Quantitative Evaluation of Systems (QEST)*, pages 157–166. IEEE Computer Society, 2006.

[93] Kim G. Larsen and Arne Skou. Bisimulation through probabilistic testing. *Information and Computation*, 94(1):1–28, 1991.

[94] Richard Lassaigne and Sylvain Peyronnet. Approximate verification of probabilistic systems. In Holger Hermanns and Roberto Segala, editors, *Process Algebra and Probabilistic Methods, Performance Modeling and Verification (PAPM/PROBMIV)*, pages 213–214. Springer, 2002.

[95] P. Lecca and C. Priami. Cell cycle control in eukaryotes: A BioSpi model. Technical Report DIT-03-045, Informatica e Telecommunicazioni: University of Trento, 2003.

[96] Tim Littlefair. CCCC web page. http://cccc.sourceforge.net/, 2008.

[97] Manish Malhotra, Jogesh K. Muppala, and Kishor S. Trivedi. Stiffness-tolerant methods for transient analysis of stiff Markov chains. *Microelectronics and Reliability*, 34(11):1825–1841, 1994.

[98] Mouad Ben Mamoun, Nihal Pekergin, and Sana Younes. Model Checking of Continuous-Time Markov Chains by Closed-Form Bounding Distributions. In *Quantitative Evaluation of Systems (QEST)*, pages 189–198. IEEE Computer Society, 2006.

[99] M. Ajmone Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis. GreatSPN web page. http://www.di.unito.it/~greatspn/index.html, 2008.

[100] Mieke Massink, Joost-Pieter Katoen, and Diego Latella. Model Checking Dependability Attributes of Wireless Group Communication. In *Dependable Systems and Networks (DSN)*, pages 711–720. IEEE Computer Society, 2004.

[101] I. Mitrani. *Simulation Techniques for Discrete Event Systems*. Cambridge University Press, New York, NY, USA, 1982.

[102] Michael Mock, Edgar Nett, and Stefan Schemmer. Efficient Reliable Real-Time Group Communication for Wireless Local Area Networks. In Jan Hlavicka, Erik Maehle, and Andrs Pataricza, editors, *European Dependable Computing Conference*, volume 1667 of *LNCS*, pages 380–400. Springer, 1999.

[103] Sri Gopal Mohanty, Aliakbar Montazer-Haghighi, and R. Trueblood. On the transient behavior of a finite birth-death process with an application. *Computers and Operations Research*, 20(3):239–248, 1993.

[104] David Monniaux. Abstract interpretation of programs as Markov decision processes. *Science of Computer Programming*, 58(1-2):179–205, 2005.

[105] MRMC: downloads. http://www.cs.utwente.nl/~zapreevis/mrmc/downloads.php, 2008.

[106] MRMC: Specifications. http://www.cs.utwente.nl/~zapreevis/mrmc/spec.html, 2008.

[107] Prism: exporting models. http://www.prismmodelchecker.org/manual /RunningPRISM/ExportingTheModel, 2008.

[108] Arthur Nadas. An extension of a theorem of Chow and Robbins on sequential confidence intervals for the mean. *Annals of Mathematical Statistics*, 40(2):667–671, 1969.

[109] G. Norman and V. Shmatikov. Analysis of probabilistic contract signing. *Journal of Computer Security*, 14(6):561–589, 2006.

[110] H.A. Oldenkamp. Probabilistic model checking: A comparison of tools. Master's thesis, University of Twente, Faculty EEMCS, Computer Science Department, Formal Methods and Tools Group, Enschede, Netherlands, 2007.

[111] D. Parker. *Implementation of Symbolic Model Checking for Probabilistic Systems.* PhD thesis, University of Birmingham, Birmingham, UK, 2002.

[112] Sergio Pissanetzky. *Sparse Matrix Technology.* Academic Press, London, UK, 1984.

[113] A. Pnueli and L. Zuck. Verification of Multiprocess Probabilistic Protocols. *Distributed Computing*, 1(1):53–72, 1986.

[114] Amir Pnueli. The Temporal Semantics of Concurrent Programs. In *Semantics of Concurrent Computation*, pages 1–20. Springer, 1979.

[115] Prism: case studies. http://www.prismmodelchecker.org/casestudies/, 2008.

[116] Prism: Workstation Cluster Example. http://www.prismmodelchecker.org /casestudies/cluster.php, 2008.

[117] GNU Project and the Free Software Foundation. GNU General Public License (GPL). http://www.gnu.org/copyleft/gpl.html, 2007.

[118] M. A. Qureshi and W. H. Sanders. A New Methodology for Calculating Distributions of Reward Accumulated During a Finite Interval. In *Fault-Tolerant Computing*, pages 116–125. IEEE Computer Society, 1996.

[119] M. K. Reiter and A. D. Rubin. Crowds: Anonymity for Web Transactions. In *ACM Transactions on Information and System Security*, volume 1, pages 66–92. ACM Press, 1998.

[120] W. H. Sanders, W. D. Obal, M. A. Qureshi, and F. K. Widjanarko. The UltraSAN modeling environment. *Performance Evaluation*, 24(1-2):89–115, 1995.

[121] Koushik Sen, Mahesh Viswanathan, and Gul Agha. Statistical Model Checking of Black-Box Probabilistic Systems. In Rajeev Alur and Doron A. Peled, editors, *Computer Aided Verification (CAV)*, volume 3114 of *LNCS*, pages 202–215. Springer, 2004.

[122] Koushik Sen, Mahesh Viswanathan, and Gul Agha. On Statistical Model Checking of Stochastic Systems. In Kousha Etessami and Sriram K. Rajamani, editors, *Computer Aided Verification (CAV)*, volume 3576 of *LNCS*, pages 266–280. Springer, 2005.

[123] Koushik Sen, Mahesh Viswanathan, and Gul Agha. VESTA: A Statistical Modelchecker and Analyzer for Probabilistic Systems. In *Quantitative Evaluation of Systems (QEST)*, page 251. IEEE Computer Society, 2005.

[124] Gerald S. Shedler. *Regenerative Stochastic Simulation.* Academic Press, London, UK, 1993.

[125] Daniel Sleator. Splay-tree implementation. http://www.link.cs.cmu.edu/splay/, 2006.

[126] Daniel Dominic Sleator and Robert Endre Tarjan. Self-adjusting binary search trees. *Journal of the ACM*, 32(3):652–686, 1985.

[127] Fabio Somenzi. CUDD: CU Decision Diagram package. http://vlsi.colorado.edu/~fabio/CUDD/, 1997. Public software.

[128] Murray Ralph Spiegel, John J. Schiller, and R. Alu Srinivasan. *Schaum's Outline of Theory and Problems of Probability and Statistics.* McGraw-Hill, New York, NY, USA, 2000.

[129] Jeremy Sproston and Susanna Donatelli. Backward Bisimulation in Markov Chain Model Checking. *IEEE Transactions on Software Engineering*, 32(8):531–546, 2006.

[130] William J. Stewart. A Comparison of Numerical Techniques in Markov Modeling. *Communications of the ACM*, 21(2):144–152, 1978.

[131] William J. Stewart. *Introduction to the Numerical Solution of Markov Chains.* Princeton University Press, Princeton, NJ, USA, 1994.

[132] Robert E. Tarjan. Depth-First Search and Linear Graph Algorithms. *SIAM Journal of Computing*, 1(2):146–160, 1972.

[133] H. C. Tijms and R. Veldman. A fast algorithm for the transient reward distribution in continuous-time Markov chains. In *Operations Research Letters*, volume 26, pages 155–158, 2000.

[134] Henk C. Tijms. *A First Course in Stochastic Models.* John Wiley & Sons, 2003.

[135] Mirco Tribastone and Stephen Gilmore. A New Generation PEPA Workbench. In *Process Algebra and Stochastically Timed Activities (PASTA)*, pages 1820–1845, 2006.

[136] Jerzy Tyszer. *Object-Oriented Computer Simulation of Discrete-Event Systems*. Kluwer Academic Publishers, Norwell, MA, USA, 1999.

[137] David A. Wheeler. SLOCCount web page. http://www.dwheeler.com/sloccount/, 2008.

[138] Ward Whitt. Continuity of generalized semi-Markov processes. *Mathematics of Operations Research*, 5:494–501, 1980.

[139] H. Younes. Black-box probabilistic verification. Technical Report CMU-CS-04-162, Carnegie Mellon University, 2004.

[140] H. Younes. *Verification and Planning for Stochastic Processes with Asynchronous Events*. PhD thesis, Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, USA, 2005.

[141] H. Younes. Ymer: A Statistical Model Checker. In Kousha Etessami and Sriram K. Rajamani, editors, *Computer Aided Verification (CAV)*, volume 3576 of *LNCS*, pages 429–433. Springer, 2005.

[142] H. Younes. Error Control for Probabilistic Model Checking. In E. Allen Emerson and Kedar S. Namjoshi, editors, *Verification, Model Checking, and Abstract Interpretation (VMCAI)*, volume 3855 of *LNCS*, pages 142–156. Springer, 2006.

[143] H. Younes, M. Kwiatkowska, G. Norman, and D. Parker. Numerical vs. Statistical Probabilistic Model Checking: An Empirical Study. In K. Jensen and A. Podelski, editors, *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 2988 of *LNCS*, pages 46–60. Springer, 2004.

[144] H. Younes and R. Simmons. Statistical Probabilistic Model Checking with a Focus on Time-Bounded Properties. *Information and Computation*, 204(9):1368–1409, 2006.

[145] Håkan Younes, Marta Kwiatkowska, Gethin Norman, and David Parker. Numerical vs. Statistical Probabilistic Model Checking. *Software Tools for Technology Transfer (STTT)*, 8(3):216–228, 2006.

[146] Håkan Younes and Reid Simmons. Probabilistic Verification of Discrete Event Systems using Acceptance Sampling. In Ed Brinksma and Kim Guldstrand Larsen, editors, *Computer Aided Verification (CAV)*, volume 2404 of *LNCS*, pages 223–235. Springer, 2002.

Titles in the IPA Dissertation Series since 2002

M.C. van Wezel. *Neural Networks for Intelligent Data Analysis: theoretical and experimental aspects.* Faculty of Mathematics and Natural Sciences, UL. 2002-01

V. Bos and J.J.T. Kleijn. *Formal Specification and Analysis of Industrial Systems.* Faculty of Mathematics and Computer Science and Faculty of Mechanical Engineering, TU/e. 2002-02

T. Kuipers. *Techniques for Understanding Legacy Software Systems.* Faculty of Natural Sciences, Mathematics and Computer Science, UvA. 2002-03

S.P. Luttik. *Choice Quantification in Process Algebra.* Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2002-04

R.J. Willemen. *School Timetable Construction: Algorithms and Complexity.* Faculty of Mathematics and Computer Science, TU/e. 2002-05

M.I.A. Stoelinga. *Alea Jacta Est: Verification of Probabilistic, Real-time and Parametric Systems.* Faculty of Science, Mathematics and Computer Science, KUN. 2002-06

N. van Vugt. *Models of Molecular Computing.* Faculty of Mathematics and Natural Sciences, UL. 2002-07

A. Fehnker. *Citius, Vilius, Melius: Guiding and Cost-Optimality in Model Checking of Timed and Hybrid Systems.* Faculty of Science, Mathematics and Computer Science, KUN. 2002-08

R. van Stee. *On-line Scheduling and Bin Packing.* Faculty of Mathematics and Natural Sciences, UL. 2002-09

D. Tauritz. *Adaptive Information Filtering: Concepts and Algorithms.* Faculty of Mathematics and Natural Sciences, UL. 2002-10

M.B. van der Zwaag. *Models and Logics for Process Algebra.* Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2002-11

J.I. den Hartog. *Probabilistic Extensions of Semantical Models.* Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2002-12

L. Moonen. *Exploring Software Systems.* Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2002-13

J.I. van Hemert. *Applying Evolutionary Computation to Constraint Satisfaction and Data Mining.* Faculty of Mathematics and Natural Sciences, UL. 2002-14

S. Andova. *Probabilistic Process Algebra.* Faculty of Mathematics and Computer Science, TU/e. 2002-15

Y.S. Usenko. *Linearization in μCRL.* Faculty of Mathematics and Computer Science, TU/e. 2002-16

J.J.D. Aerts. *Random Redundant Storage for Video on Demand.* Faculty of Mathematics and Computer Science, TU/e. 2003-01

M. de Jonge. *To Reuse or To Be Reused: Techniques for component composition and construction.* Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2003-02

J.M.W. Visser. *Generic Traversal over Typed Source Code Representations.* Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2003-03

S.M. Bohte. *Spiking Neural Networks.* Faculty of Mathematics and Natural Sciences, UL. 2003-04

T.A.C. Willemse. *Semantics and Verification in Process Algebras with Data and Timing.* Faculty of Mathematics and Computer Science, TU/e. 2003-05

S.V. Nedea. *Analysis and Simulations of Catalytic Reactions.* Faculty of Mathematics and Computer Science, TU/e. 2003-06

M.E.M. Lijding. *Real-time Scheduling of Tertiary Storage.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2003-07

H.P. Benz. *Casual Multimedia Process Annotation – CoMPAs.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2003-08

D. Distefano. *On Modelchecking the Dynamics of Object-based Software: a Foundational Approach.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2003-09

M.H. ter Beek. *Team Automata – A Formal Approach to the Modeling of Collaboration Between System Components.* Faculty of Mathematics and Natural Sciences, UL. 2003-10

D.J.P. Leijen. *The λ Abroad – A Functional Approach to Software Components.* Faculty of Mathematics and Computer Science, UU. 2003-11

W.P.A.J. Michiels. *Performance Ratios for the Differencing Method.* Faculty of Mathematics and Computer Science, TU/e. 2004-01

G.I. Jojgov. *Incomplete Proofs and Terms and Their Use in Interactive Theorem Proving.* Faculty of Mathematics and Computer Science, TU/e. 2004-02

P. Frisco. *Theory of Molecular Computing – Splicing and Membrane systems.* Faculty of Mathematics and Natural Sciences, UL. 2004-03

S. Maneth. *Models of Tree Translation.* Faculty of Mathematics and Natural Sciences, UL. 2004-04

Y. Qian. *Data Synchronization and Browsing for Home Environments.* Faculty of Mathematics and Computer Science and Faculty of Industrial Design, TU/e. 2004-05

F. Bartels. *On Generalised Coinduction and Probabilistic Specification Formats.* Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2004-06

L. Cruz-Filipe. *Constructive Real Analysis: a Type-Theoretical Formalization and Applications.* Faculty of Science, Mathematics and Computer Science, KUN. 2004-07

E.H. Gerding. *Autonomous Agents in Bargaining Games: An Evolutionary Investigation of Fundamentals, Strategies, and Business Applications.* Faculty of Technology Management, TU/e. 2004-08

N. Goga. *Control and Selection Techniques for the Automated Testing of Reactive Systems.* Faculty of Mathematics and Computer Science, TU/e. 2004-09

M. Niqui. *Formalising Exact Arithmetic: Representations, Algorithms and Proofs.* Faculty of Science, Mathematics and Computer Science, RU. 2004-10

A. Löh. *Exploring Generic Haskell.* Faculty of Mathematics and Computer Science, UU. 2004-11

I.C.M. Flinsenberg. *Route Planning Algorithms for Car Navigation.* Faculty of Mathematics and Computer Science, TU/e. 2004-12

R.J. Bril. *Real-time Scheduling for Media Processing Using Conditionally Guaranteed Budgets.* Faculty of Mathematics and Computer Science, TU/e. 2004-13

J. Pang. *Formal Verification of Distributed Systems.* Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2004-14

F. Alkemade. *Evolutionary Agent-Based Economics.* Faculty of Technology Management, TU/e. 2004-15

E.O. Dijk. *Indoor Ultrasonic Position Estimation Using a Single Base Station.* Faculty of Mathematics and Computer Science, TU/e. 2004-16

S.M. Orzan. *On Distributed Verification and Verified Distribution.* Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2004-17

M.M. Schrage. *Proxima - A Presentation-oriented Editor for Structured Documents.* Faculty of Mathematics and Computer Science, UU. 2004-18

E. Eskenazi and A. Fyukov. *Quantitative Prediction of Quality Attributes for Component-Based Software Architectures.* Faculty of Mathematics and Computer Science, TU/e. 2004-19

P.J.L. Cuijpers. *Hybrid Process Algebra.* Faculty of Mathematics and Computer Science, TU/e. 2004-20

N.J.M. van den Nieuwelaar. *Supervisory Machine Control by Predictive-Reactive Scheduling.* Faculty of Mechanical Engineering, TU/e. 2004-21

E. Ábrahám. *An Assertional Proof System for Multithreaded Java -Theory and Tool Support-* . Faculty of Mathematics and Natural Sciences, UL. 2005-01

R. Ruimerman. *Modeling and Remodeling in Bone Tissue.* Faculty of Biomedical Engineering, TU/e. 2005-02

C.N. Chong. *Experiments in Rights Control - Expression and Enforcement.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2005-03

H. Gao. *Design and Verification of Lock-free Parallel Algorithms.* Faculty of Mathematics and Computing Sciences, RUG. 2005-04

H.M.A. van Beek. *Specification and Analysis of Internet Applications.* Faculty of Mathematics and Computer Science, TU/e. 2005-05

M.T. Ionita. *Scenario-Based System Architecting - A Systematic Approach to Developing Future-Proof System Architectures.* Faculty of Mathematics and Computing Sciences, TU/e. 2005-06

G. Lenzini. *Integration of Analysis Techniques in Security and Fault-Tolerance.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2005-07

I. Kurtev. *Adaptability of Model Transformations.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2005-08

T. Wolle. *Computational Aspects of Treewidth - Lower Bounds and Network Reliability.* Faculty of Science, UU. 2005-09

O. Tveretina. *Decision Procedures for Equality Logic with Uninterpreted Functions.* Faculty of Mathematics and Computer Science, TU/e. 2005-10

A.M.L. Liekens. *Evolution of Finite Populations in Dynamic Environments.* Faculty of Biomedical Engineering, TU/e. 2005-11

J. Eggermont. *Data Mining using Genetic Programming: Classification and Symbolic Regression.* Faculty of Mathematics and Natural Sciences, UL. 2005-12

B.J. Heeren. *Top Quality Type Error Messages.* Faculty of Science, UU. 2005-13

G.F. Frehse. *Compositional Verification of Hybrid Systems using Simulation Relations.* Faculty of Science, Mathematics and Computer Science, RU. 2005-14

M.R. Mousavi. *Structuring Structural Operational Semantics.* Faculty of Mathematics and Computer Science, TU/e. 2005-15

A. Sokolova. *Coalgebraic Analysis of Probabilistic Systems.* Faculty of Mathematics and Computer Science, TU/e. 2005-16

T. Gelsema. *Effective Models for the Structure of pi-Calculus Processes with Replication.* Faculty of Mathematics and Natural Sciences, UL. 2005-17

P. Zoeteweij. *Composing Constraint Solvers.* Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2005-18

J.J. Vinju. *Analysis and Transformation of Source Code by Parsing and Rewriting.* Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2005-19

M.Valero Espada. *Modal Abstraction and Replication of Processes with Data.* Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2005-20

A. Dijkstra. *Stepping through Haskell.* Faculty of Science, UU. 2005-21

Y.W. Law. *Key management and link-layer security of wireless sensor networks: energy-efficient attack and defense.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2005-22

E. Dolstra. *The Purely Functional Software Deployment Model.* Faculty of Science, UU. 2006-01

R.J. Corin. *Analysis Models for Security Protocols.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2006-02

P.R.A. Verbaan. *The Computational Complexity of Evolving Systems.* Faculty of Science, UU. 2006-03

K.L. Man and R.R.H. Schiffelers. *Formal Specification and Analysis of Hybrid Systems.* Faculty of Mathematics and Computer Science and Faculty of Mechanical Engineering, TU/e. 2006-04

M. Kyas. *Verifying OCL Specifications of UML Models: Tool Support and Compositionality.* Faculty of Mathematics and Natural Sciences, UL. 2006-05

M. Hendriks. *Model Checking Timed Automata - Techniques and Applications.* Faculty of Science, Mathematics and Computer Science, RU. 2006-06

J. Ketema. *Böhm-Like Trees for Rewriting.* Faculty of Sciences, VUA. 2006-07

C.-B. Breunesse. *On JML: topics in tool-assisted verification of JML programs.* Faculty of Science, Mathematics and Computer Science, RU. 2006-08

B. Markvoort. *Towards Hybrid Molecular Simulations.* Faculty of Biomedical Engineering, TU/e. 2006-09

S.G.R. Nijssen. *Mining Structured Data.* Faculty of Mathematics and Natural Sciences, UL. 2006-10

G. Russello. *Separation and Adaptation of Concerns in a Shared Data Space.* Faculty of Mathematics and Computer Science, TU/e. 2006-11

L. Cheung. *Reconciling Nondeterministic and Probabilistic Choices.* Faculty of Science, Mathematics and Computer Science, RU. 2006-12

B. Badban. *Verification techniques for Extensions of Equality Logic.* Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2006-13

A.J. Mooij. *Constructive formal methods and protocol standardization.* Faculty of Mathematics and Computer Science, TU/e. 2006-14

T. Krilavičius. *Hybrid Techniques for Hybrid Systems.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2006-15

M.E. Warnier. *Language Based Security for Java and JML.* Faculty of Science, Mathematics and Computer Science, RU. 2006-16

V. Sundramoorthy. *At Home In Service Discovery.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2006-17

B. Gebremichael. *Expressivity of Timed Automata Models.* Faculty of Science, Mathematics and Computer Science, RU. 2006-18

L.C.M. van Gool. *Formalising Interface Specifications.* Faculty of Mathematics and Computer Science, TU/e. 2006-19

C.J.F. Cremers. *Scyther - Semantics and Verification of Security Protocols.* Faculty of Mathematics and Computer Science, TU/e. 2006-20

J.V. Guillen Scholten. *Mobile Channels for Exogenous Coordination of Distributed Systems: Semantics, Implementation and Composition.* Faculty of Mathematics and Natural Sciences, UL. 2006-21

H.A. de Jong. *Flexible Heterogeneous Software Systems.* Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2007-01

N.K. Kavaldjiev. *A run-time reconfigurable Network-on-Chip for streaming DSP applications.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2007-02

M. van Veelen. *Considerations on Modeling for Early Detection of Abnormalities in Locally Autonomous Distributed Systems.* Faculty of Mathematics and Computing Sciences, RUG. 2007-03

T.D. Vu. *Semantics and Applications of Process and Program Algebra.* Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2007-04

L. Brandán Briones. *Theories for Model-based Testing: Real-time and Coverage.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2007-05

I. Loeb. *Natural Deduction: Sharing by Presentation.* Faculty of Science, Mathematics and Computer Science, RU. 2007-06

M.W.A. Streppel. *Multifunctional Geometric Data Structures.* Faculty

of Mathematics and Computer Science, TU/e. 2007-07

N. Trčka. *Silent Steps in Transition Systems and Markov Chains.* Faculty of Mathematics and Computer Science, TU/e. 2007-08

R. Brinkman. *Searching in encrypted data.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2007-09

A. van Weelden. *Putting types to good use.* Faculty of Science, Mathematics and Computer Science, RU. 2007-10

J.A.R. Noppen. *Imperfect Information in Software Development Processes.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2007-11

R. Boumen. *Integration and Test plans for Complex Manufacturing Systems.* Faculty of Mechanical Engineering, TU/e. 2007-12

A.J. Wijs. *What to do Next?: Analysing and Optimising System Behaviour in Time.* Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2007-13

C.F.J. Lange. *Assessing and Improving the Quality of Modeling: A Series of Empirical Studies about the UML.* Faculty of Mathematics and Computer Science, TU/e. 2007-14

T. van der Storm. *Component-based Configuration, Integration and Delivery.* Faculty of Natural Sciences, Mathematics, and Computer Science,UvA. 2007-15

B.S. Graaf. *Model-Driven Evolution of Software Architectures.* Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2007-16

A.H.J. Mathijssen. *Logical Calculi for Reasoning with Binding.* Faculty of Mathematics and Computer Science, TU/e. 2007-17

D. Jarnikov. *QoS framework for Video Streaming in Home Networks.* Faculty of Mathematics and Computer Science, TU/e. 2007-18

M. A. Abam. *New Data Structures and Algorithms for Mobile Data.* Faculty of Mathematics and Computer Science, TU/e. 2007-19

W. Pieters. *La Volonté Machinale: Understanding the Electronic Voting Controversy.* Faculty of Science, Mathematics and Computer Science, RU. 2008-01

A.L. de Groot. *Practical Automaton Proofs in PVS.* Faculty of Science, Mathematics and Computer Science, RU. 2008-02

M. Bruntink. *Renovation of Idiomatic Crosscutting Concerns in Embedded Systems.* Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2008-03

A.M. Marin. *An Integrated System to Manage Crosscutting Concerns in Source Code.* Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2008-04

N.C.W.M. Braspenning. *Model-based Integration and Testing of High-tech Multi-disciplinary Systems.* Faculty of Mechanical Engineering, TU/e. 2008-05

M. Bravenboer. *Exercises in Free Syntax: Syntax Definition, Parsing, and Assimilation of Language Conglomerates.* Faculty of Science, UU. 2008-06

M. Torabi Dashti. *Keeping Fairness Alive: Design and Formal Verification of Optimistic Fair Exchange Protocols.* Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2008-07

I.S.M. de Jong. *Integration and Test Strategies for Complex Manufacturing Machines.* Faculty of Mechanical Engineering, TU/e. 2008-08

I. Hasuo. *Tracing Anonymity with Coalgebras*. Faculty of Science, Mathematics and Computer Science, RU. 2008-09

L.G.W.A. Cleophas. *Tree Algorithms: Two Taxonomies and a Toolkit*. Faculty of Mathematics and Computer Science, TU/e. 2008-10

I.S. Zapreev. *Model Checking Markov Chains: Techniques and Tools*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-11

Part IV

Appendices

Appendix A

Markov Reward Model Checker

A.1 Profiling MRMC with *gprof*

In this section we present performance-profiling results for MRMC obtained with *gprof*. To interpret the profile-table data one has to consider the following column notation:

- *% time* – The percentage of the total execution time the program spent in a function.
- *cumulative seconds* – The total number of seconds the computer spent executing the function, plus the time spent in functions located in the higher rows of the table.
- *self seconds* – The total number of seconds spent in this function alone. The profile listing is sorted by this number.
- *calls* – The total number of times the function was called.
- *self s/call* - The average number of seconds spent in the function per call
- *total s/call* – The average number of seconds spent in the function and its descendants per call
- *name* – The name of the function.

It is also important to note that everywhere below *gprof* was using 100 Hz sampling rate, i.e. the sampling was done every 0.01 second.

Model checking $\mathrm{S}_{<0.2}\,(busy_1 \wedge \neg serve_1)$**:**

```
%     cumulative   self              self    total
time     seconds  seconds    calls  s/call  s/call  name
31.42      43.88    43.88 73531370    0.00    0.00  multiplyUrowByConstAndAddToLUx
 9.34      56.92    13.04 70713344    0.00    0.00  get_bit_val
 7.92      67.98    11.06 31195136    0.00    0.00  set_mtx_val_ncolse
 6.25      76.71     8.73 96927744    0.00    0.00  getRoot
```

Model checking $busy_1 \implies \mathrm{P}_{\geq 1.0}\,(\Diamond poll_1)$**:**

% time	cumulative seconds	self seconds	calls	self s/call	total s/call	name
25.53	13.54	13.54	109051904	0.00	0.00	get_bit_val
19.70	23.99	10.45	31195136	0.00	0.00	set_mtx_val_ncolse
9.62	29.09	5.10	1	5.10	13.61	get_exist_until
8.21	33.45	4.36	53677099	0.00	0.00	isWithinLineDelimiter
7.47	37.41	3.96	39783435	0.00	0.00	isEndOfLineSymbol
4.17	39.62	2.21	1	2.21	16.10	read_tra_file
3.59	41.52	1.91	31195136	0.00	0.00	set_val_ncolse
3.13	43.18	1.66	3342336	0.00	0.00	scan_number
2.51	44.51	1.33	1	1.33	5.89	get_always_until

Model checking $busy_1 \implies \mathrm{P}_{\geq 0.5}\left(\Diamond^{[0,80]} poll_1\right)$**:**

% time	cumulative seconds	self seconds	calls	self s/call	total s/call	name
97.37	10064.90	10064.90	16979	0.00	0.00	multiply_mtx_cer_MV
2.16	10288.28	223.38	1	0.22	10.29	uniformization_plain
0.10	10298.81	10.53	31195136	0.00	0.00	set_mtx_val_ncolse
0.09	10308.47	9.66	75759616	0.00	0.00	get_bit_val
0.05	10313.29	4.82	1	0.00	0.01	get_exist_until

Model checking $\mathrm{P}_{\geq 0.99}\left(\Diamond^{[40,80]} serve_1\right)$**:**

% time	cumulative seconds	self seconds	calls	self s/call	total s/call	name
95.23	7696.30	7696.30	17352	0.00	0.00	multiply_mtx_cer_MV
4.09	8027.15	330.86	2	0.17	4.02	uniformization_plain
0.14	8038.31	11.16	31195136	0.00	0.00	set_mtx_val_ncolse
0.13	8048.79	10.48	85786624	0.00	0.00	get_bit_val
0.06	8053.68	4.89	1	0.00	0.01	get_exist_until

A.2 Test coverage of MRMC

The Table A.1 shows the coverage of the MRMC sources in % of tested lines as reported by *gcov*. In this table **T. lines** is the total number of code lines and **C. lines** is the number of code lines covered by tests. Also note that *lex.yy.c* and *y.tab.c* files are generated from the *io/parser/la_etmcc2.l* and *io/parser/parser_etmcc2.y* files correspondingly.

MRMC component	Source files	T. lines	C. lines	Coverage %
Command-prompt interpreter	lex.yy.c	286	105	36.71%
	y.tab.c	153	108	71.24%
	io/parser/parser_to_core.c	215	186	86.51%
		654	400	61.16%
Input-file reader	io/read_impulse_rewards.c	33	29	87.88%
	io/read_lab_file.c	34	32	94.12%
	io/read_rewards.c	10	10	100.00%
	io/read_tra_file.c	59	35	59.32%
	io/token.c	39	36	92.31%
		175	142	81.14%
Options analyzer	mcc.c	233	169	72.53%
Runtime Settings	runtime.c	223	196	87.89%
PCTL model checking	modelchecking/transient_dtmc.c	123	123	100.00%
PRCTL model checking	modelchecking/prctl.c	95	89	93.68%
	modelchecking/transient_dtmrm.c	91	91	100.00%
		186	180	96.77%
CSL model checking	modelchecking/transient_ctmc.c	345	326	94.49%
CSRL model checking	modelchecking/transient_ctmrm.c	391	353	90.28%
Common model checking	algorithms/bscc.c	153	148	96.73%
	modelchecking/steady.c	149	149	100.00%
	modelchecking/transient.c	77	45	58.44%
	modelchecking/transient_common.c	100	97	97.00%
		479	439	91.65 %
Internal Data Storage	storage/bitset.c	156	139	89.10%
	storage/kjstorage.c	53	53	100.00%
	storage/label.c	57	51	89.47%
	storage/path_graph.c	61	61	100.00%
	storage/sparse.c	479	366	76.41%
		806	670	83.13%
Bisimulation engine	lumping/lump.c	358	324	90.50%
	lumping/partition.c	122	104	85.25%
	lumping/splay.c	78	76	97.44%
		558	504	90.32%
Numerical engines	algorithms/foxglynn.c	91	83	91.21%
	algorithms/iterative_solvers.c	271	200	73.80%
		362	283	78.18%
Total coverage		4535	3785	83.46%

Table A.1: The test-suite coverage of MRMC sources

Appendix B

On-The-Fly Steady-State Detection

This section contains proofs of Chapter 3.

B.1 Fox-Glynn error bound revisited

This appendix contains proofs of Section 3.2.

Proposition 35 *For real-valued function f that does not change sign, and a Poisson density function $\gamma_i(t)$, if $\sum_{i=\mathcal{L}_\epsilon}^{\mathcal{R}_\epsilon} \gamma_i(t) \geq 1 - \frac{\varepsilon}{2}$ then the following holds:*

$$\left| \sum_{i=0}^{\infty} \gamma_i(t) f(i) - \frac{1}{W} \sum_{i=\mathcal{L}_\epsilon}^{\mathcal{R}_\epsilon} w_i(t) f(i) \right| \leq \frac{\varepsilon}{2} \cdot \|f\| \quad .$$

Proof Initially we have:

$$\sum_{i=\mathcal{L}_\epsilon}^{\mathcal{R}_\epsilon} \gamma_i(t) \geq 1 - \frac{\varepsilon}{2} \qquad \text{(B.1)}$$

$$\forall i \in \mathbb{N} : \gamma_i(t) > 0 \qquad \text{(B.2)}$$

$$\sum_{i=0}^{\infty} \gamma_i(t) = 1 \qquad \text{(B.3)}$$

$$\|f\| = \sup_{i \in \mathbb{N}} |f(i)| \qquad \text{(B.4)}$$

Let $\beta = \sum_{i=\mathcal{L}_\epsilon}^{\mathcal{R}_\epsilon} \gamma_i(t)$. We obtain from (B.3):

$$1 - \beta = \sum_{i=0}^{\mathcal{L}_\epsilon - 1} \gamma_i(t) + \sum_{i=\mathcal{R}_\epsilon + 1}^{\infty} \gamma_i(t) \qquad \text{(B.5)}$$

Using (B.1), (B.3), and (B.2) it follows $1 - \frac{\varepsilon}{2} \leq \beta \leq 1$, or equivalently:

$$-\frac{\varepsilon}{2} \leq \beta - 1 \leq 0 \qquad \text{(B.6)}$$

$$0 \leq 1 - \beta \leq \frac{\varepsilon}{2} \qquad \text{(B.7)}$$

Notice that:

$$\sum_{i=0}^{\infty}\gamma_i(t)f(i) - \frac{1}{W}\sum_{i=\mathcal{L}_\epsilon}^{\mathcal{R}_\epsilon} w_i(t)f(i) =$$
$$\underbrace{\sum_{i=0}^{\mathcal{L}_\epsilon-1}\gamma_i(t)f(i) + \sum_{i=\mathcal{R}_\epsilon+1}^{\infty}\gamma_i(t)f(i)}_{=A} + \underbrace{\sum_{i=\mathcal{L}_\epsilon}^{\mathcal{R}_\epsilon}\left(\gamma_i(t) - \frac{w_i(t)}{W}\right)f(i)}_{=B}$$

Distinguish two cases:

1. **If $\forall i \in \mathbb{N} : 0 \leq f(i) \leq \|f\|$, i.e. f is non-negative:** Then one can easily obtain:

$$0 \leq A \leq \frac{\varepsilon}{2}\cdot\|f\|, \text{ and } -\frac{\varepsilon}{2}\cdot\|f\| \leq B \leq 0, \tag{B.8}$$

   where the former inequality comes from (B.5), and (B.7). The latter inequality follows from (B.6) and the definition of β, to be more precise:

$$B = \sum_{i=\mathcal{L}_\epsilon}^{\mathcal{R}_\epsilon}\left(\gamma_i(t) - \frac{\alpha\gamma_i(t)}{\sum_{j=\mathcal{L}_\epsilon}^{\mathcal{R}_\epsilon}\alpha\gamma_j(t)}\right)f(i) =$$
$$\sum_{i=\mathcal{L}_\epsilon}^{\mathcal{R}_\epsilon}\gamma_i(t)\left(1 - \frac{1}{\sum_{j=\mathcal{L}_\epsilon}^{\mathcal{R}_\epsilon}\gamma_j(t)}\right)f(i) = \frac{\beta-1}{\beta}\sum_{i=\mathcal{L}_\epsilon}^{\mathcal{R}_\epsilon}\gamma_i(t)f(i), \text{ and}$$
$$-\frac{\varepsilon}{2}\cdot\|f\| = -\frac{\varepsilon}{2}\cdot\|f\|\cdot\frac{1}{\beta}\sum_{i=\mathcal{L}_\epsilon}^{\mathcal{R}_\epsilon}\gamma_i(t) \leq \frac{\beta-1}{\beta}\sum_{i=\mathcal{L}_\epsilon}^{\mathcal{R}_\epsilon}\gamma_i(t)f(i) \leq 0$$

   Here it is crucial that $\beta - 1 < 0$ due to (B.6) and $\frac{1}{\beta}\sum_{i=\mathcal{L}_\epsilon}^{\mathcal{R}_\epsilon}\gamma_i(t)f(i) \geq 0$ because of (B.1), (B.2).

2. **If $\forall i \in \mathbb{N} : -\|f\| \leq f(i) \leq 0$, i.e. f is non-positive:** Then symmetrically, one can easily obtain:

$$-\frac{\varepsilon}{2}\cdot\|f\| \leq A \leq 0, \text{ and } 0 \leq B \leq \frac{\varepsilon}{2}\cdot\|f\|, \tag{B.9}$$

   where the former inequality comes from (B.5) and (B.7). The latter inequality follows from (B.6) and the definition of β.

Finally, summing up inequalities in (B.8), or in (B.9) we obtain:

$$-\frac{\varepsilon}{2}\cdot\|f\| \;\leq\; \sum_{i=0}^{\infty}\gamma_i(t)f(i) - \frac{1}{W}\sum_{i=\mathcal{L}_\epsilon}^{\mathcal{R}_\epsilon} w_i(t)f(i) \;\leq\; \frac{\varepsilon}{2}\cdot\|f\|$$

□

B.2 Criteria for steady-state detection

This appendix contains proofs of Section 3.3.

B.2.1 Transient analysis

Let $p_j^{o,*}$ be the j'th component of the precise steady-state solution $\overrightarrow{p^{o,*}}$, considering forward computations, for the initial distribution $\overrightarrow{p^o}$. Let $\pi_j^{o,*}(t)$ be the j'th component of the vector $\overrightarrow{\pi^{o,*}(t)}$.

Theorem 36 *Let $(S, \mathbf{P}, L)$ be an aperiodic DTMC with initial distribution $\overrightarrow{p^o}$, steady-state distribution $\overrightarrow{p^{o,*}}$ and $Ind \subseteq S$. If for some K and $\delta > 0$ it holds that $\forall i \geq K$: $\left\|\overrightarrow{p^{o,*}} - \overrightarrow{p^o(i)}\right\|_v^\infty \leq \delta$ then for*

$$\overrightarrow{\pi^{o,*}(t)} = \sum_{i=0}^{\infty} \gamma_i(t)\overrightarrow{p^o(i)}$$

and for inaccuracy $\varepsilon > 0$:

$$\overrightarrow{\pi^o(t)} = \begin{cases} \overrightarrow{p^o(K)} & , \text{ if } K < \mathcal{L}_\epsilon \\ \frac{1}{W}\sum_{i=\mathcal{L}_\epsilon}^{K} w_i(t)\overrightarrow{p^o(i)} + \overrightarrow{p^o(K)}\left(1 - \frac{1}{W}\sum_{i=\mathcal{L}_\epsilon}^{K} w_i(t)\right) & , \text{ if } \mathcal{L}_\epsilon \leq K \leq \mathcal{R}_\epsilon \\ \frac{1}{W}\sum_{i=\mathcal{L}_\epsilon}^{\mathcal{R}_\epsilon} w_i(t)\overrightarrow{p^o(i)} & , \text{ if } K > \mathcal{R}_\epsilon \end{cases}$$

the following inequality holds:

$$\left|\sum_{j \in Ind}\left(\pi_j^{o,*}(t) - \pi_j^o(t)\right)\right| \leq 2\delta|Ind| + \frac{3}{4}\varepsilon$$

Here W, $w_i(t)$, $\mathcal{L}_\epsilon$, and $\mathcal{R}_\epsilon$ are computed using the Fox-Glynn algorithm, such that $\sum_{i=0}^{\mathcal{L}_\epsilon - 1}\gamma_i(t) \leq \frac{\varepsilon}{4}$, and $\sum_{i=\mathcal{R}_\epsilon+1}^{\infty}\gamma_i(t) \leq \frac{\varepsilon}{4}$, and $|Ind|$ is the cardinality of Ind.

Proof Since $\mathbf{P}$ is aperiodic, the steady-state distribution $\overrightarrow{p^{o,*}}$ exists. Due to the Fox-Glynn algorithm used with the refined error bound $\frac{\varepsilon}{2}$ (cf. Proposition 4), we have $w_i(t) = \alpha\gamma_i(t)$, $\gamma_i(t) = e^{-q\cdot t}\frac{(q\cdot t)^i}{i!}$, $W = \sum_{i=\mathcal{L}_\epsilon}^{\mathcal{R}_\epsilon} w_i(t)$, $\alpha \neq 0$ is some constant, and $\mathcal{L}_\epsilon$, $\mathcal{R}_\epsilon$ such that $\beta = \sum_{i=\mathcal{L}_\epsilon}^{\mathcal{R}_\epsilon}\gamma_i(t) \geq 1 - \frac{\varepsilon}{2}$. Consider now the three cases as distinguished for $\overrightarrow{\pi^o(t)}$:

1. ($K > \mathcal{R}_\epsilon$): The steady-state detection is not involved. Thus the error bound of the original Fox-Glynn method is applicable.

$$\pi_j^{o,*}(t) - \pi_j^o(t) = \sum_{i=0}^{\infty}\gamma_i(t)p_j^o(i) - \sum_{i=\mathcal{L}_\epsilon}^{\mathcal{R}_\epsilon}\frac{\gamma_i(t)}{\beta}p_j^o(i)$$

Like in the proof of Proposition 4 we get:

$$\pi_j^{o,*}(t) - \pi_j^o(t) = \underbrace{\sum_{i=\mathcal{L}_\epsilon}^{\mathcal{R}_\epsilon}\frac{\beta-1}{\beta}\gamma_i(t)p_j^o(i)}_{=A_j} + \underbrace{\sum_{i=0}^{\mathcal{L}_\epsilon-1}\gamma_i(t)p_j^o(i) + \sum_{i=\mathcal{R}_\epsilon+1}^{\infty}\gamma_i(t)p_j^o(i)}_{=B_j}$$

As the vector $\overrightarrow{p^o(i)}$ is a distribution, we have $0 \leq \sum_{j \in Ind} p_j^o(i) \leq 1$. Thus, using the initial conditions for $\sum_{i=0}^{\mathcal{L}_\epsilon - 1} \gamma_i(t)$, $\sum_{i=\mathcal{R}_\epsilon+1}^{\infty} \gamma_i(t)$ and β it easily follows that:

$$-\frac{\varepsilon}{2} \leq \sum_{j \in Ind} A_j \leq 0 \quad \text{and} \quad 0 \leq \sum_{j \in Ind} B_j \leq \frac{\varepsilon}{2}$$

Gathering the results yields:

$$\left| \sum_{j \in Ind} \left(\pi_j^{o,*}(t) - \pi_j^o(t) \right) \right| \leq \frac{\varepsilon}{2}$$

2. $(\mathcal{L}_\epsilon \leq K \leq \mathcal{R}_\epsilon)$: In this case it follows by definition:

$$\pi_j^o(t) = \frac{1}{W} \sum_{i=\mathcal{L}_\epsilon}^{K} w_i(t) p_j^o(i) + p_j^o(K) \left(1 - \frac{1}{W} \sum_{i=\mathcal{L}_\epsilon}^{K} w_i(t) \right), \quad \text{and}$$

$$\pi_j^{o,*}(t) - \pi_j^o(t) = \sum_{i=0}^{\infty} \gamma_i(t) p_j^o(i) - \sum_{i=\mathcal{L}_\epsilon}^{K} \frac{\gamma_i(t)}{\beta} p_j^o(i) - p_j^o(K) \left(1 - \sum_{i=\mathcal{L}_\epsilon}^{K} \frac{\gamma_i(t)}{\beta} \right)$$

The right-hand side of this equation can be rewritten after some standard calculations into $C_j + D_j + E_j$, where $C_j = \sum_{i=0}^{\mathcal{L}_\epsilon - 1} \gamma_i(t) p_j^o(i)$, $D_j = \sum_{i=\mathcal{L}_\epsilon}^{K} \frac{\beta - 1}{\beta} \gamma_i(t) p_j^o(i)$ and $E_j = \sum_{i=K+1}^{\infty} \gamma_i(t) p_j^o(i) - p_j^o(K) \left(1 - \sum_{i=\mathcal{L}_\epsilon}^{K} \frac{\gamma_i(t)}{\beta} \right)$.

As vector $\overrightarrow{p^o(i)}$ is a distribution, and by assumption $\sum_{i=0}^{\mathcal{L}_\epsilon} \gamma_i(t) \leq \frac{\varepsilon}{4}$:

$$0 \leq \sum_{j \in Ind} C_j \leq \frac{\varepsilon}{4}$$

From $1 - \frac{\varepsilon}{2} \leq \beta \leq 1$ and $0 \leq \sum_{j \in Ind} p_j^o(i) \leq 1$, it follows:

$$0 \geq \sum_{j \in Ind} D_j = \sum_{i=\mathcal{L}_\epsilon}^{K} \frac{\beta - 1}{\beta} \gamma_i(t) \left(\sum_{j \in Ind} p_j^o(i) \right), \text{ and}$$

$$\sum_{i=\mathcal{L}_\epsilon}^{K} \frac{\beta - 1}{\beta} \gamma_i(t) \left(\sum_{j \in Ind} p_j^o(i) \right) \geq -\frac{\varepsilon}{2\beta} \sum_{i=\mathcal{L}_\epsilon}^{K} \gamma_i(t) \geq -\frac{\varepsilon}{2}$$

After some straightforward calculations one obtains:

$$E_j = \underbrace{\sum_{i=K+1}^{\infty} \gamma_i(t) \left(p_j^o(i) - p_j^o(K) \right)}_{=F_j} \underbrace{- \sum_{i=0}^{\mathcal{L}_\epsilon - 1} \gamma_i(t) p_j^o(K)}_{=-G_j} + \underbrace{\sum_{i=\mathcal{L}_\epsilon}^{K} \frac{1 - \beta}{\beta} \gamma_i(t) p_j^o(K)}_{=H_j}$$

In a similar way as for C_j and D_j, we obtain:

$$-\frac{\varepsilon}{4} \leq \sum_{j \in Ind} G_j \leq 0, \quad \text{and} \quad 0 \leq \sum_{j \in Ind} H_j \leq \frac{\varepsilon}{2\beta} \sum_{i=\mathcal{L}_\epsilon}^{K} \gamma_i(t) \leq \frac{\varepsilon}{2}$$

To obtain bounds for F_j, we first rewrite the equation for F_j in the following way:

$$F_j = \sum_{i=K+1}^{\infty} \gamma_i(t)(p_j^o\,(i) - p_j^{o,*}) + \sum_{i=K+1}^{\infty} \gamma_i(t)(p_j^{o,*} - p_j^o\,(K))$$

From the initial condition $\forall i \geq K : \left\| \overrightarrow{p^{o,*}} - \overrightarrow{p^o\,(i)} \right\|_v^{\infty} \leq \delta$, and $\sum_{i=K+1}^{\infty} \gamma_i(t) \leq 1$ it follows:

$$-\delta \leq \sum_{i=K+1}^{\infty} \gamma_i(t)(p_j^{o,*} - p_j^o\,(K)) \leq \delta, \text{ and}$$

$$-\delta \leq \sum_{i=K+1}^{\infty} \gamma_i(t)(p_j^o\,(i) - p_j^{o,*}) \leq \delta$$

Thus $-2\delta \leq F_j \leq 2\delta$ and then it directly follows that:

$$-2\delta|Ind| \leq \sum_{j \in Ind} F_j \leq 2\delta|Ind|$$

By gathering all results, we obtain:

$$\left| \sum_{j \in Ind} \left(\pi_j^{o,*}\,(t) - \pi_j^o\,(t)\right) \right| \leq 2\delta|Ind| + \frac{3}{4}\varepsilon$$

3. $(K < \mathcal{L}_\epsilon)$: For this case, we have $\pi_j^o\,(t) = p_j^o\,(K) \sum_{i=0}^{\infty} \gamma_i(t)$ and:

$$\pi_j^{o,*}\,(t) - \pi_j^o\,(t) = \sum_{i=0}^{\infty} \gamma_i(t) \left(p_j^o\,(i) - p_j^o\,(K)\right)$$

Splitting the right-hand side of this equation yields:

$$\underbrace{\sum_{i=0}^{K} \gamma_i(t) \left(p_j^o\,(i) - p_j^o\,(K)\right)}_{=I_j} + \underbrace{\sum_{i=K+1}^{\infty} \gamma_i(t) \left(p_j^o\,(i) - p_j^o\,(K)\right)}_{=F_j}$$

Due to $K < \mathcal{L}_\epsilon$, it follows that:

$$0 \leq \sum_{j \in Ind} \sum_{i=0}^{K} \gamma_i(t) p_j^o\,(i) \leq \sum_{i=0}^{K} \gamma_i(t) \leq \frac{\varepsilon}{4}, \text{ and similarly}$$

$$0 \leq \sum_{j \in Ind} \sum_{i=0}^{K} \gamma_i(t) p_j^o\,(K) \leq \sum_{i=0}^{K} \gamma_i(t) \leq \frac{\varepsilon}{4}$$

Thus we have:

$$-\frac{\varepsilon}{4} \leq \sum_{j \in Ind} I_j \leq \frac{\varepsilon}{4}$$

For F_j we already have (cf. case 2):

$$-2\delta|Ind| \leq \sum_{j \in Ind} F_j \leq 2\delta|Ind|$$

Gathering the results yields:

$$\left| \sum_{j \in Ind} \left(\pi_j^{o,*}(t) - \pi_j^o(t) \right) \right| \leq 2\delta|Ind| + \frac{\varepsilon}{4}$$

Summarizing the results of the three proof cases, we obtain the following. For arbitrary $0 \leq K < \infty$, due to $\max\{\frac{\varepsilon}{2}, 2\delta|Ind| + \frac{3}{4}\varepsilon, 2\delta|Ind| + \frac{\varepsilon}{4}\} = 2\delta|Ind| + \frac{3}{4}\varepsilon$:

$$\left| \sum_{j \in Ind} \left(\pi_j^{o,*}(t) - \pi_j^o(t) \right) \right| \leq 2\delta|Ind| + \frac{3}{4}\varepsilon$$

□

Corollary 37 *Under the same conditions as Theorem 5:*

$$\left\| \overrightarrow{p^{o,*}} - \overrightarrow{p^o(K)} \right\|_v^\infty \leq \frac{\varepsilon}{8|Ind|} \quad \textit{implies} \quad \left| \sum_{j \in Ind} \left(\pi_j^{o,*}(t) - \pi_j^o(t) \right) \right| \leq \varepsilon$$

Proof According to Theorem 5 if $\left\| \overrightarrow{p^{o,*}} - \overrightarrow{p^o(K)} \right\|_v^\infty \leq \delta$ then:

$$\left| \sum_{j \in Ind} \left(\pi_j^{o,*}(t) - \pi_j^o(t) \right) \right| \leq 2\delta|Ind| + \frac{3}{4}\varepsilon$$

By taking $\delta = \frac{\varepsilon}{8|Ind|}$, we have:

$$\left| \sum_{j \in Ind} \left(\pi_j^{o,*}(t) - \pi_j^o(t) \right) \right| \leq \frac{2\varepsilon|Ind|}{8|Ind|} + \frac{3}{4}\varepsilon = \varepsilon$$

□

B.2.2 Backward computations

Let $\pi_j^*(t)$ be the j'th component of $\overrightarrow{\pi^*(t)}$, and p_j^* be the j'th component of $\overrightarrow{p^*}$.

Theorem 38 *Let $(S, \mathbf{P}, L)$ be an aperiodic DTMC with $Ind \subseteq S$ such that $\forall j \in Ind$: $P(j,j) = 1$, $\overrightarrow{p(i)} = \mathbf{P}^i \cdot \overrightarrow{1_{Ind}}$ and steady-state vector $\overrightarrow{p^*}$. If for some K and $\delta > 0$ it holds that $\forall i \geq K : \forall j \in \mathbb{N}_{[1,N]} : 0 \leq p_j^* - p_j(i) \leq \delta$, then for*

$$\overrightarrow{\pi^*(t)} = \sum_{i=0}^{\infty} \gamma_i(t) \overrightarrow{p(i)}$$

and for inaccuracy $\varepsilon > 0$:

$$\overrightarrow{\pi\,(t)} = \begin{cases} \overrightarrow{p\,(K)} & , \text{ if } K < \mathcal{L}_\epsilon \\ \frac{1}{W}\sum_{i=\mathcal{L}_\epsilon}^{K} w_i(t)\overrightarrow{p\,(i)} + \overrightarrow{p\,(K)}\left(1 - \frac{1}{W}\sum_{i=\mathcal{L}_\epsilon}^{K} w_i(t)\right) & , \text{ if } \mathcal{L}_\epsilon \le K \le \mathcal{R}_\epsilon \\ \frac{1}{W}\sum_{i=\mathcal{L}_\epsilon}^{\mathcal{R}_\epsilon} w_i(t)\overrightarrow{p\,(i)} & , \text{ if } K > \mathcal{R}_\epsilon \end{cases}$$

the following inequality holds:

$$\boxed{\left\|\overrightarrow{\pi^*\,(t)} - \overrightarrow{\pi\,(t)}\right\|_v^{\infty} \le \delta + \frac{3}{4}\varepsilon}$$

Here W, $w_i(t)$, $\mathcal{L}_\epsilon$, and $\mathcal{R}_\epsilon$ are computed using the Fox-Glynn algorithm, such that $\sum_{i=0}^{\mathcal{L}_\epsilon - 1}\gamma_i(t) \le \frac{\varepsilon}{4}$, and $\sum_{i=\mathcal{R}_\epsilon+1}^{\infty}\gamma_i(t) \le \frac{\varepsilon}{4}$.

Proof Since $\mathbf{P}$ is aperiodic, the steady-state vector $\overrightarrow{p^*}$ exists. Due to the Fox-Glynn algorithm used with the refined desired error bound $\frac{\varepsilon}{2}$ (cf. Proposition 4), we have $w_i(t) = \alpha\gamma_i(t)$, $\gamma_i(t) = e^{-q\cdot t}\frac{(q\cdot t)^i}{i!}$, $W = \sum_{i=\mathcal{L}_\epsilon}^{\mathcal{R}_\epsilon} w_i(t)$, $\alpha \neq 0$ is some constant, and $\mathcal{L}_\epsilon$, $\mathcal{R}_\epsilon$ such that $\beta = \sum_{i=\mathcal{L}_\epsilon}^{\mathcal{R}_\epsilon}\gamma_i(t) \ge 1 - \frac{\varepsilon}{2}$.
Consider now the three cases as distinguished for $\overrightarrow{\pi\,(t)}$:

1. $(K > \mathcal{R}_\epsilon)$: The steady-state detection is not involved. Thus the error bound of the original Fox-Glynn method is applicable.

$$\pi_j^*(t) - \pi_j(t) = \sum_{i=0}^{\infty}\gamma_i(t)p_j(i) - \sum_{i=\mathcal{L}_\epsilon}^{\mathcal{R}_\epsilon}\frac{\gamma_i(t)}{\beta}p_j(i)$$

   Like in the proof of Proposition 4 we get:

$$\pi_j^*(t) - \pi_j(t) = \underbrace{\sum_{i=0}^{\mathcal{L}_\epsilon-1}\gamma_i(t)p_j(i) + \sum_{i=\mathcal{R}_\epsilon+1}^{\infty}\gamma_i(t)p_j(i)}_{=A_j} + \underbrace{\sum_{i=\mathcal{L}_\epsilon}^{\mathcal{R}_\epsilon}\frac{\beta-1}{\beta}\gamma_i(t)p_j(i)}_{=B_j}$$

   The vector $\overrightarrow{p\,(i)}$ is such that $0 \le p_j(i) \le 1$. Using the initial conditions for $\sum_{i=0}^{\mathcal{L}_\epsilon-1}\gamma_i(t)$, $\sum_{i=\mathcal{R}_\epsilon+1}^{\infty}\gamma_i(t)$ and β it easily follows that:

$$0 \le A_j \le \frac{\varepsilon}{2} \quad \text{and} \quad -\frac{\varepsilon}{2} \le B_j \le 0$$

   Gathering the results yields:

$$\left|\pi_j^*(t) - \pi_j(t)\right| \le \frac{\varepsilon}{2} \tag{B.10}$$

2. $(\mathcal{L}_\epsilon \le K \le \mathcal{R}_\epsilon)$: In this case it follows by definition:

$$\pi_j(t) = \frac{1}{W}\sum_{i=\mathcal{L}_\epsilon}^{K} w_i(t)p_j(i) + p_j(K)\left(1 - \frac{1}{W}\sum_{i=\mathcal{L}_\epsilon}^{K} w_i(t)\right), \text{ and}$$

$$\pi_j^*(t) - \pi_j(t) = \sum_{i=0}^{\infty} \gamma_i(t) p_j(i) - \sum_{i=\mathcal{L}_\epsilon}^{K} \frac{\gamma_i(t)}{\beta} p_j(i) - p_j(K) \left(1 - \sum_{i=\mathcal{L}_\epsilon}^{K} \frac{\gamma_i(t)}{\beta}\right)$$

The right-hand side of this equation can be rewritten after some standard calculations into $C_j + D_j + E_j$, where $C_j = \sum_{i=0}^{\mathcal{L}_\epsilon - 1} \gamma_i(t) p_j(i)$, $D_j = \sum_{i=\mathcal{L}_\epsilon}^{K} \frac{\beta-1}{\beta} \gamma_i(t) p_j(i)$ and $E_j = \sum_{i=K+1}^{\infty} \gamma_i(t) p_j(i) - p_j(K) \left(1 - \sum_{i=\mathcal{L}_\epsilon}^{K} \frac{\gamma_i(t)}{\beta}\right)$.

From the fact that $0 \leq p_j(i) \leq 1$, and by assumption $\sum_{i=0}^{\mathcal{L}_\epsilon - 1} \gamma_i(t) \leq \frac{\varepsilon}{4}$:

$$0 \leq C_j \leq \frac{\varepsilon}{4}$$

From $1 - \frac{\varepsilon}{2} \leq \beta \leq 1$ and $0 \leq p_j(i) \leq 1$, it follows:

$$-\frac{\varepsilon}{2} \leq -\frac{\varepsilon}{2\beta} \sum_{i=\mathcal{L}_\epsilon}^{K} \gamma_i(t) \leq \sum_{i=\mathcal{L}_\epsilon}^{K} \frac{\beta-1}{\beta} \gamma_i(t) = D_j \leq 0$$

After some straightforward computations one obtains:

$$E_j = \underbrace{\sum_{i=K+1}^{\infty} \gamma_i(t) \left(p_j(i) - p_j(K)\right)}_{=F_j} \underbrace{- \sum_{i=0}^{\mathcal{L}_\epsilon - 1} \gamma_i(t) p_j(K)}_{=-G_j} + \underbrace{\sum_{i=\mathcal{L}_\epsilon}^{K} \frac{1-\beta}{\beta} \gamma_i(t) p_j(K)}_{H_j} \tag{B.11}$$

In a similar way as for C_j and D_j, we obtain:

$$-\frac{\varepsilon}{4} \leq G_j \leq 0, \quad \text{and} \quad 0 \leq H_j \leq \frac{\varepsilon}{2\beta} \sum_{i=\mathcal{L}_\epsilon}^{K} \gamma_i(t) \leq \frac{\varepsilon}{2}$$

To derive bounds for F_j, we first rewrite the equation for F_j in the following way:

$$F_j = \sum_{i=K+1}^{\infty} \gamma_i(t)(p_j(i) - p_j^*) + \sum_{i=K+1}^{\infty} \gamma_i(t)(p_j^* - p_j(K))$$

From the initial condition $\forall i \geq K : \forall j \in \mathbb{N}_{[1,N]} : 0 \leq p_j^* - p_j(i) \leq \delta$, and $0 \leq p_j(i) \leq 1$ it follows:

$$0 \leq \sum_{i=K+1}^{\infty} \gamma_i(t)(p_j^* - p_j(K)) \leq \delta,$$

and because of the fact that probabilities $p_j(i)$ are not decreasing, due to the initial condition $\forall j \in Ind : P(j,j) = 1$:

$$-\delta \leq \sum_{i=K+1}^{\infty} \gamma_i(t)(p_j(i) - p_j^*) \leq 0$$

From which we conclude that $-\delta \leq F_j \leq \delta$. Then, by gathering all results, we obtain:

$$\left|\pi_j^*(t) - \pi_j(t)\right| \leq \delta + \frac{3}{4}\varepsilon$$

3. $(K < \mathcal{L}_\epsilon)$: For this case, we have:

$$\pi_j(t) = p_j(K) \sum_{i=0}^{\infty} \gamma_i(t), \quad \text{and} \quad \pi_j^*(t) - \pi_j(t) = \sum_{i=0}^{\infty} \gamma_i(t) \left(p_j(i) - p_j(K)\right)$$

Splitting the right-hand side of this equation yields:

$$\underbrace{\sum_{i=0}^{K} \gamma_i(t) \left(p_j(i) - p_j(K)\right)}_{=I_j} + \underbrace{\sum_{i=K+1}^{\infty} \gamma_i(t) \left(p_j(i) - p_j(K)\right)}_{=F_j}$$

Due to $K < \mathcal{L}_\epsilon$, it follows that:

$$0 \leq \sum_{i=0}^{K} \gamma_i(t) p_j(i) \leq \frac{\varepsilon}{4}, \text{ and } 0 \leq \sum_{i=0}^{K} \gamma_i(t) p_j(K) \leq \frac{\varepsilon}{4}$$

Thus we have $-\frac{\varepsilon}{4} \leq I_j \leq \frac{\varepsilon}{4}$ and for F_j(cf. case 2) $-\delta \leq F_j \leq \delta$. Gathering the results yields:

$$\left|\pi_j^*(t) - \pi_j(t)\right| \leq \delta + \frac{\varepsilon}{4}$$

Summarizing the results of the three proof cases, we obtain the following. For arbitrary $0 \leq K < \infty$ and any $j \in \mathbb{N}_{[1,N]}$, due to $\max\{\frac{\varepsilon}{2}, \delta + \frac{3}{4}\varepsilon, \delta + \frac{\varepsilon}{4}\} = \delta + \frac{3}{4}\varepsilon$:

$$\left|\pi_j^*(t) - \pi_j(t)\right| \leq \delta + \frac{3}{4}\varepsilon, \quad \text{that implies} \quad \left\|\overrightarrow{\pi^*(t)} - \overrightarrow{\pi(t)}\right\|_v^\infty \leq \delta + \frac{3}{4}\varepsilon.$$

□

Corollary 39 *Under the same conditions as Theorem 7:*

$$\left\|\overrightarrow{p^*} - \overrightarrow{p(K)}\right\|_v^\infty \leq \frac{\varepsilon}{4} \textit{ implies } \left\|\overrightarrow{\pi^*(t)} - \overrightarrow{\pi(t)}\right\|_v^\infty \leq \varepsilon$$

Proof According to Theorem 7 if $\left\|\overrightarrow{p^*} - \overrightarrow{p(K)}\right\|_v^\infty \leq \delta$ then:

$$\left|\pi_j^*(t) - \pi_j(t)\right| \leq \delta + \frac{3}{4}\varepsilon$$

By taking $\delta = \frac{\varepsilon}{4}$, we have for any $j \in \mathbb{N}_{[1,N]}$:

$$\left|\pi_j^*(t) - \pi_j(t)\right| \leq \frac{\varepsilon}{4} + \frac{3}{4}\varepsilon = \varepsilon, \quad \text{that implies} \quad \left\|\overrightarrow{\pi^*(t)} - \overrightarrow{\pi(t)}\right\|_v^\infty \leq \varepsilon.$$

□

B.3 Safely detecting stationarity

This appendix contains proofs of Section 3.4.

Proposition 40 *For any state s in CTMC $(S,\ \mathbf{Q},\ L)$, time-bounded property $\mathcal{A}\ \mathrm{U}^{[0,t]}\ \mathcal{G}$ and $\mathbf{Q}^B = \mathbf{Q}\,[\mathcal{I}\cup\mathcal{G}]\,[B_{\mathcal{A},\mathcal{G}}]$ we have:*

$$Prob\left(s,\ \mathcal{A}\ \mathrm{U}^{[0,t]}\ \mathcal{G}\right)\ in\ (S,\ \mathbf{Q},\ L)\ =\ Prob\left(s,\ S\ \mathrm{U}^{[t,t]}\ \mathcal{G}\right)\ in\ (S,\mathbf{Q}^B)$$

Proof In [8] the following is proved:

$$Prob\left(s,\ \mathcal{A}\ \mathrm{U}^{[0,t]}\ \mathcal{G}\right)\ \text{in}\ (S,\ \mathbf{Q},\ L)\ =\ Prob\left(s,\ S\ \mathrm{U}^{[t,t]}\ \mathcal{G}\right)\ \text{in}\ (S,\mathbf{Q}\,[\mathcal{I}\cup\mathcal{G}])$$

It is also clear that

$$Prob\left(s,\ S\ \mathrm{U}^{[t,t]}\ \mathcal{G}\right)\ \text{in}\ (S,\mathbf{Q}\,[\mathcal{I}\cup\mathcal{G}])\ =\ Prob\left(s,\ S\ \mathrm{U}^{[t,t]}\ \mathcal{G}\right)\ \text{in}\ (S,\mathbf{Q}^B)$$

The latter is due to the fact, that for a BSCC B in $\mathbf{Q}\,[\mathcal{I}\cup\mathcal{G}]$:

$$\text{if}\ \exists s_1\in B: s_1\in\mathcal{A}\setminus\mathcal{G}\ \text{then}\ \forall s_2\in B: s_2\in\mathcal{A}\setminus\mathcal{G}$$

and thus from any state $s_1\in B_{\mathcal{A},\mathcal{G}}$ it is impossible to reach $\mathcal{G}$. □

Theorem 41 *For the stochastic matrix $\mathbf{P}_B$ obtained after uniformizing CTMC $(S,\ \mathbf{Q}^B)$, for any K and $\delta>0$ the following holds:*

$$\sum_{j\in\mathcal{A}\setminus(\mathcal{G}\cup B_{\mathcal{A},\mathcal{G}})} p^s_j(K)\le\delta\Rightarrow\forall i\ge K:\left\|\overrightarrow{p^{s,*}}-\overrightarrow{p^s(i)}\right\|_v^{\infty}\le\delta$$

Where $p^s_j(i)$ is the j'th component of $\overrightarrow{p^s(i)}=\overrightarrow{1_{\{s\}}}\cdot(\mathbf{P}_B)^i$, and $\overrightarrow{p^{s,*}}$ is the steady-state probability for $\mathbf{P}_B$ when starting from state s.

Proof In [81] it was noticed that in $\mathbf{Q}\,[\mathcal{I}\cup\mathcal{G}]$ all $\mathcal{G}$ states can be collapsed into one state, without affecting $Prob\left(s,\ S\ \mathrm{U}^{[t,t]}\ \mathcal{G}\right)$. The same can be done with the $\mathcal{I}$ states. In $\mathbf{Q}^B$ the $B_{\mathcal{A},\mathcal{G}}$ states are also made absorbing, as this does not affect $\overrightarrow{p^s(i)}$. Thus, as a trivial extension, we suggest to collapse all $B_{\mathcal{A},\mathcal{G}}\cup\mathcal{I}$ states of $\mathbf{Q}^B$ into a single absorbing state. This yields a matrix, denoted $\mathbf{Q}^B$, with two absorbing states, one that corresponds to $\mathcal{G}$ states - say state N, and one that corresponds to $B_{\mathcal{A},\mathcal{G}}\cup\mathcal{I}$ states - say state $N-1$. Here N denotes the number of states that result after the described procedure. The remaining states $N-2$ are transient states from the set $\mathcal{A}\setminus(\mathcal{G}\cup B_{\mathcal{A},\mathcal{G}})=\{1,\cdots,N-2\}$.

The rest of this proof is divided into three steps:

1. First, let us prove that for any K and $\delta>0$:

$$\sum_{j\in\mathbb{N}_{[1,N-2]}} p^s_j(K)\le\delta\Rightarrow\left\|\overrightarrow{p^{s,*}}-\overrightarrow{p^s(K)}\right\|_v^{\infty}\le\delta \tag{B.12}$$

   By definition of the l^{∞}-norm:

$$\left\|\overrightarrow{p^{s,*}}-\overrightarrow{p^s(K)}\right\|_v^{\infty}=\max_{j\in\mathbb{N}_{[1,N]}}|p^{s,*}_j-p^s_j(K)| \tag{B.13}$$

Since states $1, \cdots, N-2$ are transient $\forall j \in \mathbb{N}_{[1,N-2]} : p_j^{s,*} = 0$, and thus (B.13) equals:

$$\max\left\{\max_{j\in\mathbb{N}_{[1,N-2]}} p_j^s(K), \max_{j\in\{N-1,N\}} |p_j^{s,*} - p_j^s(K)|\right\} \tag{B.14}$$

Using $\sum_{j\in\mathbb{N}_{[1,N-2]}} p_j^s(K) \leq \delta$ we get that (B.14) is bounded from above by:

$$\max\left\{\delta, \max_{j\in\{N-1,N\}} |p_j^{s,*} - p_j^s(K)|\right\} \tag{B.15}$$

Vectors $\overrightarrow{p^s(K)}$ and $\overrightarrow{p^{s,*}}$ are distributions:

$$\sum_{j=1}^{N-2} p_j^s(K) + p_{N-1}^s(K) + p_N^s(K) = 1 \tag{B.16}$$

$$p_{N-1}^{s,*} + p_N^{s,*} = 1 \tag{B.17}$$

From (B.16) and (B.17) it follows:

$$p_{N-1}^{s,*} - p_{N-1}^s(K) + p_N^{s,*} - p_N^s(K) = \sum_{j=1}^{N-2} p_j^s(K)$$

As the probability mass is flowing into the $\mathcal{G}$, $\mathcal{I}$ and $B_{\mathcal{A},\mathcal{G}}$ states, we have:

$$0 \leq p_{N-1}^{s,*} - p_{N-1}^s(K) \text{ and } 0 \leq p_N^{s,*} - p_N^s(K)$$

and thus:

$$|p_{N-1}^{s,*} - p_{N-1}^s(K)| + |p_N^{s,*} - p_N^s(K)| = \sum_{j=1}^{N-2} p_j^s(K)$$

From the latter and the initial condition $\sum_{j=1}^{N-2} p_j^s(K) \leq \delta$ we get:

$$|p_{N-1}^{s,*} - p_{N-1}^s(K)| + |p_N^{s,*} - p_N^s(K)| \leq \delta$$

which induces:

$$|p_{N-1}^{s,*} - p_{N-1}^s(K)| \leq \delta \text{ and } |p_N^{s,*} - p_N^s(K)| \leq \delta$$

Finally, it follows that (B.15) is limited from above by:

$$max\{\delta, \delta, \delta\} = \delta$$

which yields (B.12).

2. The next step is to prove that for any K:

$$\forall Z > 0 : \sum_{j\in\mathbb{N}_{[1,N-2]}} p_j^s(K) \geq \sum_{j\in\mathbb{N}_{[1,N-2]}} p_j^s(K+Z) \tag{B.18}$$

The latter clearly follows from the fact that for any K:

$$\sum_{j\in\mathbb{N}_{[1,N-2]}} p_j^s(K) \geq \sum_{j\in\mathbb{N}_{[1,N-2]}} p_j^s(K{+}1) \tag{B.19}$$

Equation (B.19) follows from the fact that states $N-1$ and N are absorbing states in $\mathbf{P}_B = \left(p_{i,j}^B\right)$, in other words:

$$p_{N-1}^s(K+1) = \underbrace{\sum_{j\in\mathbb{N}_{[1,N-2]}} p_j^s(K)\cdot p_{j,N-1}^B}_{\geq 0} + p_{N-1}^s(K) \tag{B.20}$$

$$p_N^s(K+1) = \underbrace{\sum_{j\in\mathbb{N}_{[1,N-2]}} p_j^s(K)\cdot p_{j,N}^B}_{\geq 0} + p_N^s(K) \tag{B.21}$$

Vectors $\overrightarrow{p^s(K)}$ and $\overrightarrow{p^s(K{+}1)}$ are distributions, thus:

$$\sum_{j\in\mathbb{N}_{[1,N-2]}} p_j^s(K) - \sum_{j\in\mathbb{N}_{[1,N-2]}} p_j^s(K{+}1) =$$
$$\left(1{-}p_{N-1}^s(K) - p_N^s(K)\right) - \left(1{-}p_{N-1}^s(K+1) - p_N^s(K+1)\right) \tag{B.22}$$

From (B.20), (B.21) and (B.22) we obtain:

$$\sum_{j\in\mathbb{N}_{[1,N-2]}} p_j^s(K) - \sum_{j\in\mathbb{N}_{[1,N-2]}} p_j^s(K{+}1) =$$
$$\underbrace{p_{N-1}^s(K+1) - p_{N-1}^s(K)}_{\geq 0} + \underbrace{p_N^s(K+1) - p_N^s(K)}_{\geq 0} \geq 0$$

which yields (B.19).

3. The last step is to notice that, due to (B.18), for any K and $\delta > 0$:

$$\sum_{j\in\mathbb{N}_{[1,N-2]}} p_j^s(K) \leq \delta \Rightarrow \forall i \geq K : \sum_{j\in\mathbb{N}_{[1,N-2]}} p_j^s(i) \leq \delta$$

and from (B.12) for any i:

$$\sum_{j\in\mathbb{N}_{[1,N-2]}} p_j^s(i) \leq \delta \Rightarrow \left\|\overrightarrow{p^{s,*}} - \overrightarrow{p^s(i)}\right\|_v^\infty \leq \delta$$

This proves the claim.

□

Theorem 42 *For the stochastic matrix $\mathbf{P}_B$ obtained after uniformizing CTMC $(S,\ \mathbf{Q}^B)$, for any K and $\delta > 0$ the following holds:*

$$\left\|\overrightarrow{1} - \left(\overrightarrow{p(K)} + \overrightarrow{p^B(K)}\right)\right\|_v^\infty \leq \delta \Rightarrow \forall i \geq K : \left\|\overrightarrow{p^*} - \overrightarrow{p(i)}\right\|_v^\infty \leq \delta \tag{B.23}$$

where $\overrightarrow{p(i)} = (\mathbf{P}_B)^i \cdot \overrightarrow{1_{\mathcal{G}}}$, $\overrightarrow{p^B(i)} = (\mathbf{P}_B)^i \cdot \overrightarrow{1_{B_{\mathcal{A},\mathcal{G}\cup\mathcal{I}}}}$, and $\overrightarrow{p^*} = \lim_{i\to\infty}(\mathbf{P}_B)^i \cdot \overrightarrow{1_{\mathcal{G}}}$.

Proof Consider the j'th component of vectors in (B.23), then follow the proof of Theorem 10, taking into account that:

$$1 - \left(p_j(i) + p_j^B(i)\right) = \sum_{k \in \mathcal{A} \setminus (\mathcal{G} \cup B_{\mathcal{A},\mathcal{G}})} p_k^j(i)$$

□

Appendix C

Model Checking by Discrete Event Simulation

This section contains proofs of the theorems from Chapter 6.

C.1 Unbounded-until operator

This Appendix contains proofs for Section 6.2.

Proposition 43 *For any $N \in \mathbb{N}$ the inequality $A_l \leq \alpha_g \leq A_r$ holds for any:*

$$A_l \in \left\{\alpha_g^N,\ 1-\left(\alpha_b^N+\alpha_t^N\right),\ 1-\alpha_{b,t}^N\right\} \quad \textit{and} \quad A_r \in \left\{\alpha_{g,t}^N,\ \alpha_g^N+\alpha_t^N,\ 1-\alpha_b^N\right\}.$$

Proof According to Proposition 18 for any N we have:

$$\alpha_g^N+\alpha_b^N+\alpha_t^N=1, \quad \text{and} \quad \alpha_g+\alpha_b=1.$$

Here the probability α_t^N is non-increasing with the increase of N and the probabilities α_g^N and α_b^N are non-decreasing with the increase of N. We also have $\lim\limits_{N\to\infty}\left(\alpha_t^N\right)=0$, $\lim\limits_{N\to\infty}\left(\alpha_g^N\right)=\alpha_g$ and $\lim\limits_{N\to\infty}\left(\alpha_b^N\right)=\alpha_b$, where the first limit holds because in the long run the probability to be in a transient state is zero.

Clearly, considering any epoch N in the limit the probability mass α_t^N is distributed between α_g and α_b. Therefore we can conclude that:

$$\alpha_g^N \leq \alpha_g \leq \alpha_g^N+\alpha_t^N.$$

From Proposition 18 it follows that:

$$\alpha_g^N=1-\left(\alpha_b^N+\alpha_t^N\right)=1-\alpha_{b,t}^N, \quad \text{and} \quad \alpha_g^N+\alpha_t^N=\alpha_{g,t}^N=1-\alpha_b^N,$$

and this concludes the proof. □

The following lemmas are purely technical but they are necessary for several proofs of this Appendix.

Lemma 44 *For any $k \in \mathcal{N}$, let Γ_N^k be computed on a sample $\overrightarrow{\mathbf{P}_N} = \left(\mathbf{P}_N^1, \ldots, \mathbf{P}_N^M\right)$ of $M \in \mathbb{N}_{\geq 1}$ observations, then the following holds:*

$$\Gamma_N^{g,b,t} = \Gamma_N^g + \Gamma_N^b + \Gamma_N^t = M, \tag{C.1}$$

$$\Gamma_N^{g,t} = \Gamma_N^g + \Gamma_N^t, \tag{C.2}$$

$$\Gamma_N^{b,t} = \Gamma_N^b + \Gamma_N^t. \tag{C.3}$$

Proof Let us sketch the proofs of the given equalities.

Considering Definition 19 it is clear that Γ_N^k represents the number of "k"-type states in the sample $\overrightarrow{\mathbf{P}_N}$ of size M. The simulated Markov chain $\mathbf{P}^B$ has only three disjoint sets of states: "good", "bad" and "transient" states, which are "counted" by Γ_N^g, Γ_N^b and Γ_N^t correspondingly. The latter implies that Equation (C.1) holds.

The proof of Equation (C.2) is trivial since $f_{g,t}\left(\mathbf{P}_N\right) = f_g\left(\mathbf{P}_N\right) + f_t\left(\mathbf{P}_N\right)$. Similarly, we can prove Equation (C.3). □

Lemma 45 *For any $k \in \mathcal{N}$, let Γ_N^k be computed on a sample $\overrightarrow{\mathbf{P}_N} = \left(\mathbf{P}_N^1, \ldots, \mathbf{P}_N^M\right)$ of $M \in \mathbb{N}_{\geq 1}$ observations, then the following holds:*

$$\overline{X}_g^N + \overline{X}_b^N + \overline{X}_t^N = 1, \quad \overline{X}_{g,t}^N = \overline{X}_g^N + \overline{X}_t^N, \quad \overline{X}_{b,t}^N = \overline{X}_b^N + \overline{X}_t^N.$$

Proof This is a trivial consequence of Lemma 44 and Equation 6.5. □

Lemma 46 *For a sample $\overrightarrow{\mathbf{P}_N} = \left(\mathbf{P}_N^1, \ldots, \mathbf{P}_N^M\right)$ of $M \in \mathbb{N}_{\geq 1}$ independent observations, $X_i = f_k\left(\mathbf{P}_N^i\right)$, with $k \in \mathcal{N}$, and $\overline{X}_k^N$ with $\overline{V}_k^N$ computed by Equations (5.1) and (5.7) correspondingly, the following holds:*

$$\overline{X}_k^N = \frac{\Gamma_N^k}{M}, \overline{V}_k^N = \sqrt{\frac{1}{M-1}\sum_{i=1}^{M}\left(f_k\left(\mathbf{P}_N^i\right) - \frac{\Gamma_N^k}{M}\right)^2} =$$

$$= \sqrt{\frac{\Gamma_N^k \cdot \left(M - \Gamma_N^k\right)}{M \cdot (M-1)}} = \sqrt{\frac{\overline{X}_k^N \cdot \left(M - \Gamma_N^k\right)}{M-1}}.$$

Proof The case of $\overline{X}_k^N$ trivially follows from Definition 19 and Equation (5.1).

Let us consider $\overline{V}_k^N$:

$$\overline{V}_k^N = \sqrt{\frac{1}{M-1}\sum_{i=1}^{M}\left(f_k\left(\mathbf{P}_N^i\right)-\overline{X}_k^N\right)^2} =$$

$$= \sqrt{\frac{\mathbf{1}}{\mathbf{M-1}}\sum_{\mathbf{i=1}}^{\mathbf{M}}\left(\mathbf{f_k}\left(\mathbf{P_N^i}\right)-\frac{\mathbf{\Gamma_N^k}}{\mathbf{M}}\right)^{\mathbf{2}}} =$$

$$= \sqrt{\frac{1}{M-1}\left(\Gamma_N^k\cdot\left(1-\frac{\Gamma_N^k}{M}\right)^2+\left(M-\Gamma_N^k\right)\cdot\left(\frac{\Gamma_N^k}{M}\right)^2\right)} =$$

$$= \sqrt{\frac{1}{M^2\cdot(M-1)}\left(\Gamma_N^k\cdot\left(M-\Gamma_N^k\right)^2+\left(M-\Gamma_N^k\right)\cdot\left(\Gamma_N^k\right)^2\right)} =$$

$$= \sqrt{\frac{\mathbf{\Gamma_N^k}\cdot\left(\mathbf{M}-\mathbf{\Gamma_N^k}\right)}{\mathbf{M}\cdot(\mathbf{M}-\mathbf{1})}} = \sqrt{\frac{\mathbf{\overline{X}_k^N}\cdot\left(\mathbf{M}-\mathbf{\Gamma_N^k}\right)}{\mathbf{M}-\mathbf{1}}}, \tag{C.4}$$

here the needed representations are emphasized with the bold font. □

C.1.1 Dependency of the confidence intervals

In this section we give proofs of the dependency between certain confidence intervals in cases of finite sample size M and $M \to \infty$.

The following density functions are going to be used. They are an obvious consequence of the way α_k^N is defined for any $k \in \mathcal{N}$:

$$Prob\left(f_k\left(\mathbf{P}_N\right)=i\right) = \begin{cases} \alpha_k^N & \textbf{iff } i=1 \\ 1-\alpha_k^N & \textbf{iff } i=0 \end{cases}, \tag{C.5}$$

for all $k,l \in \{g,b,t\},\ k \neq l$:

$$Prob\left(f_k\left(\mathbf{P}_N\right)=i, f_l\left(\mathbf{P}_N\right)=j\right) = \begin{cases} \alpha_k^N & \textbf{iff } i=1 \wedge j=0 \\ \alpha_l^N & \textbf{iff } i=0 \wedge j=1 \\ 0 & \textbf{iff } i=1 \wedge j=1 \\ 1-\left(\alpha_k^N+\alpha_l^N\right) & \textbf{iff } i=0 \wedge j=0 \end{cases}, \tag{C.6}$$

and two more joint-density functions:

$$Prob\left(f_g\left(\mathbf{P}_N\right)=i, f_{g,t}\left(\mathbf{P}_N\right)=j\right) = \begin{cases} 0 & \textbf{iff } i=1 \wedge j=0 \\ \alpha_t^N & \textbf{iff } i=0 \wedge j=1 \\ \alpha_g^N & \textbf{iff } i=1 \wedge j=1 \\ 1-\alpha_{g,t}^N & \textbf{iff } i=0 \wedge j=0 \end{cases}, \tag{C.7}$$

$$Prob\left(f_b\left(\mathbf{P}_N\right)=i, f_{b,t}\left(\mathbf{P}_N\right)=j\right) = \begin{cases} 0 & \textbf{iff } i=1 \wedge j=0 \\ \alpha_t^N & \textbf{iff } i=0 \wedge j=1 \\ \alpha_b^N & \textbf{iff } i=1 \wedge j=1 \\ 1-\alpha_{b,t}^N & \textbf{iff } i=0 \wedge j=0 \end{cases}. \tag{C.8}$$

Below we are going to prove the dependency of the *c. i.* in the above-mentioned cases.

Case: finite $M \in \mathbb{N}_{\geq 2}$.

Let us consider the following technical lemmas.

Lemma 47 *For any constants $a \in \mathbb{N}_{\geq 2}$ and $b \in \mathbb{R}_{[0,1]}$ the function:*

$$\widetilde{\Upsilon}_{a,b}(x) = \frac{x - a \cdot b}{\sqrt{\frac{x \cdot (a-x)}{a-1}}}, \tag{C.9}$$

is continuous and increasing on $x \in \mathbb{R}_{(0,a)}$.

Proof In order to show that $\widetilde{\Upsilon}_{a,b}(x)$ is continuous on $\mathbb{R}_{(0,a)}$ it is enough to show that it has the first derivative. Differentiating $\widetilde{\Upsilon}_{a,b}(x)$ by x gives us:

$$\widetilde{\Upsilon}'_{a,b}(x) = \frac{a}{2 \cdot \sqrt{\frac{x \cdot (a-x)}{a-1}}} \cdot \frac{(1 - 2 \cdot b) \cdot x + a \cdot b}{x \cdot (a - x)}. \tag{C.10}$$

that takes finite values for all $x \in \mathbb{R}_{(0,a)}$.

In order to prove that $\widetilde{\Upsilon}_{a,b}(x)$ is increasing on $\mathbb{R}_{(0,a)}$ it is enough to show that $\widetilde{\Upsilon}'_{a,b}(x)$ is strictly positive on this interval. Then, considering Equation (C.10), it suffices to prove that:

$$(1 - 2 \cdot b) \cdot x + a \cdot b > 0 \tag{C.11}$$

for all values of x, a and b, as stated in the conditions of this lemma. Note that $x \cdot (a - x) > 0$ for $x \in \mathbb{R}_{(0,a)}$, and $\sqrt{\frac{x \cdot (a-x)}{a-1}}$ is taken with the positive sign.

First, let us analyze when Equation (C.11) turns into zero. In fact it happens only for $x_1 = \frac{a \cdot b}{2 \cdot b - 1}$ under the condition $b \neq \frac{1}{2}$, because $a \geq 2$. It is easy to see that $x_1 \notin \mathbb{R}_{(0,a)}$ for any $a \in \mathbb{N}_{\geq 2}$ and $b \in \mathbb{R}_{[0,1]} \setminus \left\{\frac{1}{2}\right\}$, consider the two cases:

1. If $b \in \mathbb{R}_{\left[0,\frac{1}{2}\right)}$ then clearly $x_1 < 0$.
2. If $b \in \mathbb{R}_{\left(\frac{1}{2},1\right]}$ then $x_1 > 0$, but we need to have $x_1 < a$. Which is equivalent to $a < a \cdot b$. The latter does not hold for the given choice of b.

Since Equation (C.11) defines a continuous function which does not turn into zero on $\mathbb{R}_{(0,a)}$, then it is either positive or negative on this interval. Thus, to find out the sign of the function, it is enough to sample it on one combination of parameters. For example, taking $b = 1$ and $x = a - 1$ turns Equation (C.11) into $1 > 0$. Therefore $\widetilde{\Upsilon}_{a,b}(x)$ is an increasing function on $\mathbb{R}_{(0,a)}$. □

Lemma 48 *For any $k \in \mathcal{N}$, the* c. i. *of α_k^N, given by Equation (6.2) is defined by the* r. v. $\Upsilon_k\left(\Gamma_N^k\right) = \widetilde{\Upsilon}_{M,\alpha_k^N}\left(\Gamma_N^k\right)$*, where $\widetilde{\Upsilon}_{a,b}(x)$ is given by Equation (C.9) of Lemma 47.*

Proof The *c. i.* of α_k^N, see Equation (6.2), is based on the *r. v.* defined in Equation (5.3) with the unknown variance σ substituted by the sample variance V, provided by Equation (5.7), and the point estimate $\overline{X}$ computed by Equation (5.1). Therefore, for the case of α_k^N, the *r. v.* defined by Equation (5.3) looks as follows:

$$\frac{\overline{X}_k^N - \alpha_k^N}{\overline{V}_k^N / \sqrt{M}}, \tag{C.12}$$

and defines the *c. i.* of α_k^N in the following way:

$$Prob\left(-\widetilde{z_n}\left(\beta\right) \leq \frac{\overline{X}_k^N - \alpha_k^N}{\overline{V}_k^N/\sqrt{M}} \leq \widetilde{z_n}\left(\beta\right)\right) \approx 1-\beta. \tag{C.13}$$

The latter is a trivial statement, since Equation (6.2), together with Equation (6.4), is equivalent to Equation (C.13).

The the formula (that defines a *r. v.*) given by Equation (C.12) can obviously be rewritten, using Definition 19 and Lemma 46, as a function $\Upsilon_k\left(\Gamma_N^k\right) = \widetilde{\Upsilon}_{M,\alpha_k^N}\left(\Gamma_N^k\right)$ of Γ_N^k. □

Lemma 49 *Let $\alpha_k^N \in \mathbb{R}_{(0,1)}$ for any $k \in \mathcal{N}$, $\overrightarrow{\mathbf{P}_N} = \left(\mathbf{P}_N^1, \ldots, \mathbf{P}_N^M\right)$ is a sample of $M \in \mathbb{N}_{\geq 2}$ independent observations and $\Gamma_N^k \in \mathbb{N}_{[0,M]}$ then $\Upsilon_k\left(\Gamma_N^k\right) \in \mathbb{R}$* **iff** *$\Gamma_N^k \in \mathbb{N}_{(0,M)}$, $\lim\limits_{\Gamma_N^k \to 0}\left(\Upsilon_k\left(\Gamma_N^k\right)\right) = -\infty$ and $\lim\limits_{\Gamma_N^k \to M}\left(\Upsilon_k\left(\Gamma_N^k\right)\right) = +\infty$.*

Proof Clearly, by definition of Γ_N^k we have $\Gamma_N^k \in \mathbb{N}_{[0,M]}$, and by Lemma 48:

$$\Upsilon_k\left(\Gamma_N^k\right) = \frac{\Gamma_N^k - M \cdot \alpha_k^N}{\sqrt{\frac{\Gamma_N^k \cdot \left(M - \Gamma_N^k\right)}{M-1}}},$$

where $M - 1 > 0$, because $M \in \mathbb{N}_{\geq 2}$.

Since, $\alpha_k^N \in \mathbb{R}_{(0,1)}$, then obviously $\lim\limits_{\Gamma_N^k \to 0}\left(\Upsilon_k\left(\Gamma_N^k\right)\right) = -\infty$ and $\lim\limits_{\Gamma_N^k \to M}\left(\Upsilon_k\left(\Gamma_N^k\right)\right) = +\infty$. For $\Gamma_N^k \in \mathbb{N}_{(0,M)}$ function $\Upsilon_k\left(\Gamma_N^k\right)$ clearly takes finite real values. □

Further we will make no distinction between $+\infty$ and $-\infty$, in both cases we will use just ∞. Moreover, for technical reasons, we extend function $\Upsilon_k\left(\Gamma_N^k\right)$ (by continuity) and write $\Upsilon_k\left(\Gamma_N^k\right) = \infty$ for $\Gamma_N^k \in \{0, M\}$.

Lemma 50 *Let $\alpha_k^N \in \mathbb{R}_{(0,1)}$ for any $k \in \mathcal{N}$, then for a sample $\overrightarrow{\mathbf{P}_N} = \left(\mathbf{P}_N^1, \ldots, \mathbf{P}_N^M\right)$ of $M \in \mathbb{N}_{\geq 2}$ independent observations:*

$$Prob\left(\Upsilon_k\left(\Gamma_N^k\right) = A\right) = \begin{cases} Prob\left(\Gamma_N^k = 0\right) + Prob\left(\Gamma_N^k = M\right) & , \quad A = \infty \\ Prob\left(\Gamma_N^k = C\right) & , \quad A < \infty \wedge A = \Upsilon_k\left(C\right) \end{cases} \tag{C.14}$$

Note that, since $\Gamma_N^k \in \mathbb{N}_{[0,M]}$ is a discrete r. v.*, then $Prob\left(\Gamma_N^k = C\right) = 0$ if $C \notin \mathbb{N}_{[0,M]}$.*

Proof Lemma 49 implies that for $\alpha_k^N \notin \{0,1\}$ the value of $\Upsilon_k\left(\Gamma_N^k\right)$ is infinite in case of $\Gamma_N^k \in \{0, M\}$ and is a finite real value for $\Gamma_N^k \in \mathbb{R}_{(0,M)}$.

The former case implies:

$$Prob\left(\Upsilon_k\left(\Gamma_N^k\right) = \infty\right) = Prob\left(\Gamma_N^k = 0 \vee \Gamma_N^k = M\right) = Prob\left(\Gamma_N^k = 0\right) + Prob\left(\Gamma_N^k = M\right),$$

where the last equality is because the events $\Gamma_N^k = 0$ and $\Gamma_N^k = M$ are disjoint, i. e. they can not hold simultaneously.

In the latter case, Lemma 47 is applicable because for all $k \in \mathcal{N}$ we have $\alpha_k^N \in \mathbb{R}_{[0,1]}$. By this lemma $\widetilde{\Upsilon}_{a,b}\left(x\right)$ is an increasing function on $\mathbb{R}_{(0,a)}$, implying that $\forall x, y \in: x \neq$

$y \Leftrightarrow \widetilde{\Upsilon}_{a,b}(x) \neq \widetilde{\Upsilon}_{a,b}(y)$. Therefore, $\Upsilon_k\left(\Gamma_N^k\right)$ is finite and injective on $\mathbb{R}_{(0,M)}$, meaning that:

$$Prob\left(\Upsilon_k\left(\Gamma_N^k\right) = A\right) = Prob\left(\Gamma_N^k = C\right)$$

with $A < \infty$ and C such that $A = \Upsilon_k(C)$. □

Lemma 51 *For a sample* $\overrightarrow{\mathbf{P}_N} = \left(\mathbf{P}_N^1, \ldots, \mathbf{P}_N^M\right)$ *of* $M \in \mathbb{N}_{\geq 1}$ *independent observations, seen as a sequence of* r.v.*, and any* $k \in \mathcal{N}$*:*

$$Prob\left(\Gamma_N^k = C\right) = \begin{cases} \binom{M}{C} \cdot \left(\alpha_k^N\right)^C \cdot \left(1 - \alpha_k^N\right)^{M-C} & , \quad C \in \mathbb{N}_{[0,M]} \\ 0 & , \quad else \end{cases}. \tag{C.15}$$

Proof First, let us notice that $\binom{M}{C}$ is a binomial coefficient, giving the number of different combinations a sub sample of size C can be chosen from a sample of size M. Considering $\overrightarrow{\mathbf{P}_N}$ as a sequence of *r. v.* we have that Γ_N^k is a sum of M *i. i. d. r. v.* According to Equation (C.5), these *r. v.* take value 1 with probability α_k^N and 0 with probability $1 - \alpha_k^N$. Then the fact that $\Gamma_N^k = C$ with $C \in \mathbb{N}_{[0,M]}$ means that C *r. v.* take values 1 and $M - C$ *r. v.* take values 0. The probability of such an event is $\left(1 - \alpha_k^N\right)^{M-C} \cdot \left(\alpha_k^N\right)^C$, plus there are $\binom{M}{C}$ possible ways the *r. v.* that take value 1 can be distributed in the sample.

In case $C \notin \mathbb{N}_{[0,M]}$ the probability $Prob\left(\Gamma_N^k = C\right)$ is clearly zero. □

Lemma 52 *Let* $\alpha_k^N \in \mathbb{R}_{(0,1)}$ *for any* $k \in \mathcal{N}$*, then for a sample* $\overrightarrow{\mathbf{P}_N} = \left(\mathbf{P}_N^1, \ldots, \mathbf{P}_N^M\right)$ *of* $M \in \mathbb{N}_{\geq 2}$ *independent observations, and any* $k, l \in \mathcal{N}$ *such that* $k \neq l$*:*

$$Prob\left(\Upsilon_k\left(\Gamma_N^k\right) = A, \quad \Upsilon_l\left(\Gamma_N^l\right) = B\right) =$$

$$= \begin{cases} Prob\left(\Gamma_N^k = 0, \quad \Gamma_N^l = 0\right) + Prob\left(\Gamma_N^k = 0, \quad \Gamma_N^l = M\right) + \\ Prob\left(\Gamma_N^k = M, \quad \Gamma_N^l = 0\right) + Prob\left(\Gamma_N^k = M, \quad \Gamma_N^l = M\right) & , \quad A = \infty, \quad B = \infty \\ Prob\left(\Gamma_N^k = C, \quad \Gamma_N^l = 0\right) + Prob\left(\Gamma_N^k = C, \quad \Gamma_N^l = M\right) & , \quad A < \infty, \quad B = \infty \\ Prob\left(\Gamma_N^k = 0, \quad \Gamma_N^l = D\right) + Prob\left(\Gamma_N^k = M, \quad \Gamma_N^l = D\right) & , \quad A = \infty, \quad B < \infty \\ Prob\left(\Gamma_N^l = C, \quad \Gamma_N^l = D\right) & , \quad A < \infty, \quad B < \infty \end{cases}$$

with C *and* D *such that* $A = \Upsilon_k(C)$ *and* $B = \Upsilon_l(D)$*. Note that, since* $\Gamma_N^k, \Gamma_N^l \in \mathbb{N}_{[0,M]}$ *are discrete* r.v.*, then, for example,* $Prob\left(\Gamma_N^l = C, \quad \Gamma_N^l = D\right) = 0$ *if* $C \notin \mathbb{N}_{[0,M]}$*.*

Proof Similar to the proof of Lemma 50, this proof is solely based on Lemma 49 and Lemma 47. Let us consider the following cases:

- $A = \infty, \quad B = \infty$**:** For any $k \in \mathcal{N}$ we have $\Upsilon_k\left(\Gamma_N^k\right) = \infty$ **iff** $\Gamma_N^k = 0$ or $\Gamma_N^k = M$, where the events $\Gamma_N^k = 0$ and $\Gamma_N^k = M$ are disjoint. The latter means that they can not hold simultaneously. Therefore we have:

$$\begin{aligned} Prob\left(\Upsilon_k\left(\Gamma_N^k\right) = A, \quad \Upsilon_l\left(\Gamma_N^l\right) = B\right) = \\ Prob\left(\Gamma_N^k = 0 \vee \Gamma_N^k = M, \quad \Gamma_N^l = 0 \vee \Gamma_N^l = M\right) = \\ = Prob\left(\Gamma_N^k = 0, \quad \Gamma_N^l = 0 \vee \Gamma_N^l = M\right) + \\ Prob\left(\Gamma_N^k = M, \quad \Gamma_N^l = 0 \vee \Gamma_N^l = M\right) = \\ Prob\left(\Gamma_N^k = 0, \quad \Gamma_N^l = 0\right) + Prob\left(\Gamma_N^k = 0, \quad \Gamma_N^l = M\right) + \\ Prob\left(\Gamma_N^k = M, \quad \Gamma_N^l = 0\right) + Prob\left(\Gamma_N^k = M, \quad \Gamma_N^l = M\right). \end{aligned}$$

- $A < \infty, \quad B = \infty$**:** There exists a unique $C \in \mathbb{R}_{(0,M)}$, with $A = \Upsilon_k(C)$, and $\Upsilon_l\left(\Gamma_N^l\right) = B$ **iff** $\Gamma_N^l = 0$ or $\Gamma_N^l = M$. The latter events are disjoint and thus:

$$\begin{aligned} Prob\left(\Upsilon_k\left(\Gamma_N^k\right) = A, \quad \Upsilon_l\left(\Gamma_N^l\right) = B\right) &= Prob\left(\Gamma_N^k = C, \quad \Gamma_N^l = 0 \vee \Gamma_N^l = M\right) = \\ &= Prob\left(\Gamma_N^k = C, \quad \Gamma_N^l = 0\right) + Prob\left(\Gamma_N^k = C, \quad \Gamma_N^l = M\right). \end{aligned}$$

  The case of $A = \infty, \quad B < \infty$ is symmetric to this one.

- $A < \infty, \quad B < \infty$**:** There exist unique $C, D \in \mathbb{R}_{(0,M)}$, with $A = \Upsilon_k(C)$ and $B = \Upsilon_l(D)$, and thus:

$$Prob\left(\Upsilon_k\left(\Gamma_N^k\right) = A, \quad \Upsilon_l\left(\Gamma_N^l\right) = B\right) = Prob\left(\Gamma_N^k = C, \quad \Gamma_N^l = D\right).$$

□

Lemma 53 *For a sample* $\overrightarrow{\mathbf{P}_N} = \left(\mathbf{P}_N^1, \ldots, \mathbf{P}_N^M\right)$ *of* $M \in \mathbb{N}_{\geq 1}$ *independent observations, seen as a sequence of* r.v.*, and any* $k, l \in \mathcal{N}$ *such that* $k \neq l$*:*

- *if* $l \cap k = \emptyset$ *then:*

$$Prob\left(\Gamma_N^k = C,\ \Gamma_N^l = D\right) = \left\{ \begin{array}{lll} P_1 & , & C, D \in \mathbb{N}_{[0,M]}, \quad C + D \leq M \\ 0 & , & else \end{array} \right. , \quad \text{(C.16)}$$

- *if* $k \subset l$ *then:*

$$Prob\left(\Gamma_N^k = C,\ \Gamma_N^l = D\right) = \left\{ \begin{array}{lll} P_2 & , & C, D \in \mathbb{N}_{[0,M]}, \quad C \leq D \\ 0 & , & else \end{array} \right. , \quad \text{(C.17)}$$

- *if* $k \cap l \neq \emptyset \wedge k \not\subset l \wedge l \not\subset k$ *then:*

$$Prob\left(\Gamma_N^k = C,\ \Gamma_N^l = D\right) = \left\{ \begin{array}{lll} P_3 & , & C, D \in \mathbb{N}_{[0,M]}, \quad M \leq D + C \\ 0 & , & else \end{array} \right. , \quad \text{(C.18)}$$

where the values of P_1*,* P_2 *and* P_3 *are defined as follows:*

$$\begin{aligned} P_1 &= \binom{M}{C} \cdot \binom{M-C}{D} \cdot \left(\alpha_k^N\right)^C \cdot \left(\alpha_l^N\right)^D \cdot \left(1 - \alpha_{k\cup l}^N\right)^{M-(C+D)}, \\ P_2 &= \binom{M}{C} \cdot \binom{M-C}{D-C} \cdot \left(\alpha_k^N\right)^C \cdot \left(\alpha_{l\setminus k}^N\right)^{D-C} \cdot \left(1 - \alpha_l^N\right)^{M-D}, \\ P_3 &= \binom{M}{M-D} \cdot \binom{D}{M-C} \cdot \left(\alpha_{k\setminus l}^N\right)^{M-D} \cdot \left(\alpha_{l\setminus k}^N\right)^{M-C} \cdot \left(\alpha_{k\cap l}^N\right)^{C+D-M}. \end{aligned}$$

Proof Clearly, for all $k \in \mathcal{N}$ we have $\Gamma_N^k \in \mathbb{N}_{[0,M]}$ and therefore unless $C, D \in \mathbb{N}_{[0,M]}$ the above mentioned probabilities turn into zero.

Let us consider the following two cases:

- $k \cap l = \emptyset$**:** The condition implies that $\forall i \in \mathbb{N}_{[1,M]} : \neg \left(f_k \left(\mathbf{P}_N^i \right) = 1 \wedge f_l \left(\mathbf{P}_N^i \right) = 1 \right)$, meaning that Γ_N^k and Γ_N^l "count" different kind of states in the sample. Now, by Lemma 44 we have $\Gamma_N^k + \Gamma_N^l \leq M$, which implies that for $C + D > M$:

$$Prob\left(\Gamma_N^k = C,\ \Gamma_N^l = D\right) = 0.$$

For $C + D \leq M$, in a sample of M observations we have $\binom{M}{C}$ possible sub-samples with states of type k, for each of which, in the remaining part of the sample, we have $\binom{M-C}{D}$ possible sub-samples with states of type l. In total we get $\binom{M}{C} \cdot \binom{M-C}{D}$ unique possibilities to have a sample with C states of type k and D states of type l. Clearly the probability of each such sample is computed as:

$$\left(\alpha_k^N\right)^C \cdot \left(\alpha_l^N\right)^D \cdot \left(1 - \alpha_{k \cup l}^N\right)^{M-(C+D)},$$

since $\alpha_{k\cup l}^N = \alpha_k^N + \alpha_l^N$ by Proposition 18 and this concludes the proof of Equation (C.16).

- $k \subset l$**:** The condition implies that there can be Γ_N^k states of type k in the sample and thus, by Lemma 44, $\Gamma_N^l - \Gamma_N^k$ states of kind $l \setminus k$. The latter implies that unless $C \leq D$ then:

$$Prob\left(\Gamma_N^k = C,\ \Gamma_N^l = D\right) = 0.$$

The rest of the proof goes similar to the previous case, considering that $k \cap (l \setminus k) = \emptyset$ and $\alpha_k^N + \alpha_{l\setminus k}^N = \alpha_l^N$ by Proposition 18.

- $k \cap l \neq \emptyset \wedge k \not\subset l \wedge l \not\subset k$**:** Let $m = k \cap l$, then the condition implies that $\Gamma_N^k = \Gamma_N^{k\setminus m} + \Gamma_N^m$ and $\Gamma_N^l = \Gamma_N^{l\setminus m} + \Gamma_N^m$.
By definition $\mathcal{N} = \{\{g\}, \{b\}, \{t\}, \{g,t\}, \{b,t\}\}$, therefore we either have $k = \{b,t\}$ and $l = \{g,t\}$ or $k = \{g,t\}$ and $l = \{b,t\}$. Both cases are symmetric and thus let us choose the latter one making $m = \{t\}$, $k \setminus m = \{g\}$ and $l \setminus m = \{b\}$.

From Lemma 44 it follows that $M \leq \Gamma_N^{g,t} + \Gamma_N^{b,t}$, implying that unless $M \leq C + D$ we have:

$$Prob\left(\Gamma_N^{g,t} = C,\ \Gamma_N^{b,t} = D\right) = 0.$$

For the remaining cases, since $\Gamma_N^t \in \mathbb{N}_{[0,M]}$ we can state that:

$$Prob\left(\Gamma_N^{g,t} = C,\ \Gamma_N^{b,t} = D\right) = \sum_{i=0}^{M} Prob\left(\Gamma_N^{g,t} = C,\ \Gamma_N^{b,t} = D,\ \Gamma_N^t = i\right).$$

Notice that from Lemma 44 it follows that we have $\Gamma_N^t = \Gamma_N^{g,t} + \Gamma_N^{b,t} - M$, which means that for all $i \neq C + D - M$:

$$Prob\left(\Gamma_N^{g,t} = C,\ \Gamma_N^{b,t} = D,\ \Gamma_N^t = i\right) = 0.$$

The latter implies that:

$$\begin{aligned} &Prob\left(\Gamma_N^{g,t} = C,\ \Gamma_N^{b,t} = D\right) = \\ &= Prob\left(\Gamma_N^g = M - D,\ \Gamma_N^b = M - C,\ \Gamma_N^t = C + D - M\right), \end{aligned}$$

which similarly to the previous cases can be expressed as:

$$Prob\left(\Gamma_N^{g,t}=C,\ \Gamma_N^{b,t}=D\right)=\binom{M}{M-D}\cdot\binom{D}{M-C}\cdot\left(\alpha_g^N\right)^{M-D}\cdot\left(\alpha_b^N\right)^{M-C}\cdot\left(\alpha_t^N\right)^{C+D-M}$$

□

Proposition 54 *Let* $\overrightarrow{\mathbf{P}_N}=\left(\mathbf{P}_N^1,\ldots,\mathbf{P}_N^M\right)$ *be a sample of* $M\in\mathbb{N}_{\geq 3}$ *independent observations. For any* $k,l\in\mathcal{N}$, $k\neq l$ *and* $\alpha_k^N,\alpha_l^N\notin\{0,1\}$ *the* c. i. *of* α_k^N *and* α_l^N[1] *are dependent.*

Proof By Lemma 48, for $k,l\in\mathcal{N}$ the *c. i.* of α_k^N and α_l^N are dependent **iff** *r. v.* $\Upsilon_k\left(\Gamma_N^k\right)$, $\Upsilon_l\left(\Gamma_N^l\right)$ are dependent. Which is also equivalent to showing that:

$$Prob\left(\Upsilon_k\left(\Gamma_N^k\right)=A,\quad\Upsilon_l\left(\Gamma_N^l\right)=B\right)\neq Prob\left(\Upsilon_k\left(\Gamma_N^k\right)=A\right)\cdot Prob\left(\Upsilon_l\left(\Gamma_N^l\right)=B\right),\tag{C.19}$$

with some $A,B\in\mathbb{R}\cup\{\infty\}$, for the joint density function of $\Upsilon_k\left(\Gamma_N^k\right)$ and $\Upsilon_l\left(\Gamma_N^l\right)$.

We are going to show that Equation (C.19) holds by switching from *r. v.* $\Upsilon_k\left(\Gamma_N^k\right)$ to Γ_N^k, as provided by Lemmas 50, 52, and using the density functions given by Lemmas 51, 53.

In the subsequent part of this proof we are not considering $A=\infty$ or $B=\infty$, because it corresponds to the case when the sample variance $\overline{V}_k^N$ or $\overline{V}_l^N$ turns into zero and the *c. i.* degrade into single points. The applicability of Central Limit Theorem itself in this case could be questioned, since it requires the true value of variances σ_k^N and σ_l^N to be different from zero.

By Lemma 49 for any $C,D\in\mathbb{N}_{(0,M)}$ there exist $A=\Upsilon_k(C)$, $B=\Upsilon_l(D)$ such that $A,B<\infty$. Therefore showing that Equation (C.19) holds for some $A,B<\infty$ can be done by showing that:

$$Prob\left(\Gamma_N^k=C,\quad\Gamma_N^l=D\right)\neq Prob\left(\Gamma_N^k=C\right)\cdot Prob\left(\Gamma_N^l=D\right),\tag{C.20}$$

holds for some $C,D\in\mathbb{N}_{(0,M)}$. Note that Equation (C.20) follows Equation (C.19) with the help of Lemma 50 and Lemma 52 for $A=\Upsilon_k(C)$, $B=\Upsilon_l(D)$ and $A,B<\infty$.

Since $|\mathcal{N}|=5$ and we have to prove the pairwise dependency of the *c. i.*, consider the following ten cases:

1. **The *c. i.* of α_g^N and α_t^N:** For finding C and D such that Equation (C.20) holds we are going to employ Lemma 51 and Lemma 53.

   First, notice that $\{g\}\cap\{t\}=\emptyset$ and since $M\geq 3$ we can always choose $C,D\in\mathbb{N}_{(0,M)}: C+D>M$. For such C and D, Equation (C.20) holds because by Lemma 53 its left-hand side turns into zero, where as its right-hand side is some non-zero value. The latter is according to Lemma 51 and the fact that for all $k\in\mathcal{N}:\alpha_k^N\notin\{0,1\}$.

2. **The *c. i.* of α_b^N and α_t^N:** Similar to the case 1, since $\{b\}\cap\{t\}=\emptyset$.

3. **The *c. i.* of α_g^N and α_b^N:** Similar to the case 1, since $\{g\}\cap\{b\}=\emptyset$.

[1] Derived using Equation (6.2).

4. **The *c. i.* of α_b^N and $\alpha_{g,t}^N$:** Similar to the case 1, since $\{b\} \cap \{g,t\} = \emptyset$.
5. **The *c. i.* of α_g^N and $\alpha_{b,t}^N$:** Similar to the case 1, since $\{g\} \cap \{b,t\} = \emptyset$.
6. **The *c. i.* of α_g^N and $\alpha_{g,t}^N$:** Similar to the case 1, but notice that $\{g\} \subset \{g,t\}$ and, since $M \geq 3$, we can always choose $C, D \in \mathbb{N}_{(0,M)} : D < C$.
7. **The *c. i.* of α_t^N and $\alpha_{g,t}^N$:** Similar to the case 6, since $\{t\} \subset \{g,t\}$.
8. **The *c. i.* of α_b^N and $\alpha_{b,t}^N$:** Similar to the case 6, since $\{b\} \subset \{b,t\}$.
9. **The *c. i.* of α_t^N and $\alpha_{b,t}^N$:** Similar to the case 6, since $\{t\} \subset \{b,t\}$.
10. **The *c. i.* of $\alpha_{g,t}^N$ and $\alpha_{b,t}^N$:** Similar to the case 1, but notice that $\{g,t\} \cap \{b,t\} \neq \emptyset$, $\{g,t\} \not\subset \{b,t\}$ and $\{b,t\} \not\subset \{g,t\}$. Then since $M \geq 3$, we can always choose $C, D \in \mathbb{N}_{(0,M)} : C + D < M$.

□

Case: $M \to \infty$.

In order to prove dependency between the *c. i.* for $M \to \infty$, we consider the covariance of the limiting *r. v.* that define these *c. i.* Doing this we assume that the distribution of $f_k(\mathbf{P}_N)$, for any $k \in \mathcal{N}$, is known, allowing us to compute the covariance matrices, and to apply the multi-dimensional Central Limit Theorem.

First, let us consider the following definitions:

Definition 24 *Let Z_i, $1 \leq i \leq K$, be independent random variables with standard normal distribution. Then $\overrightarrow{Z} \cdot \mathcal{B} + \overrightarrow{W}$ is the* K-dimensional random vector with normal distribution*, where $\overrightarrow{Z} = (Z_1, \ldots, Z_K)$, $\mathcal{B} = (b_{i,j})$, $1 \leq i, j \leq K$ is a $K \times K$ matrix, and $\overrightarrow{W} = (W_1, \ldots, W_K)$ is a K-dimensional vector.*

Definition 25 *A probability measure on the measurable sets of the K-dimensional Euclidean space $\mathbb{R}^K$ is called a* K-dimensional normal distribution **iff** *it equals the distribution of a K-dimensional random vector with normal distribution.*

Note that the K-dimensional normal distribution is uniquely determined by its mean value $\overrightarrow{E}$ and covariance matrix Σ, with a density function:

$$\mathcal{F}\left(\overrightarrow{Z}\right) = \frac{1}{(2\pi)^{K/2} \mid \Sigma \mid^{1/2}} exp\left(-\frac{1}{2}\left(\overrightarrow{Z} - \overrightarrow{E}\right)\Sigma^{-1}\left(\overrightarrow{Z} - \overrightarrow{E}\right)^T\right) \tag{C.21}$$

where $\mid \Sigma \mid$ is the determinant of Σ. To relate the K-dimensional normal distribution to Definition 24 we should admit that $\Sigma = \mathcal{B} \cdot \mathcal{B}^T$ and $\overrightarrow{E} = \overrightarrow{W}$.

Definition 26 *An m-dimensional normal distribution is called* centered *if and only if $\overrightarrow{E} = (0, \ldots, 0)$.*

Theorem 55 Multi-dimensional Central Limit Theorem *[18] For $i = 1, 2, \ldots$ let $\overrightarrow{Z_i} = (Z_{1,i}, \ldots, Z_{K,i})$ be a sequence of* i. i. d. *random vectors. Suppose that $E\left[Z_{k,i}^2\right] < \infty$, then let $\overrightarrow{E} = (E[Z_{1,i}], \ldots, E[Z_{K,i}])$, and let $\Sigma = (\sigma_{n,m})$ be a covariance matrix for $\overrightarrow{Z_i}$ with $\sigma_{n,m} = Cov[Z_{n,i},\ Z_{m,i}]$. Then for $M \to \infty$:*

$$\frac{\sum_{i=1}^{M} \overrightarrow{Z_i} - M \cdot \overrightarrow{E}}{\sqrt{M}}$$

converges in distribution to the centered normal distribution with covariance matrix Σ.

In order to apply the multi-dimensional Central Limit Theorem for showing the dependency of the *r. v.* defining the *c. i.*, it is necessary to compute covariance matrices for the pairs of random variables $f_k(\mathbf{P}_N)$ and $f_l(\mathbf{P}_N)$ where $k, l \in \mathcal{N}$ and $k \neq l$.

Lemma 56 *Consider* r. v. $f_k(\mathbf{P}_N)$ *and* $f_l(\mathbf{P}_N)$*, with $k, l \in \mathcal{N}$, then:*

$$Cov[f_k(\mathbf{P}_N),\ f_l(\mathbf{P}_N)] = E[f_k(\mathbf{P}_N) \cdot f_l(\mathbf{P}_N)] - \alpha_k^N \cdot \alpha_l^N, \tag{C.22}$$

$$Var[f_k(\mathbf{P}_N)] = \alpha_k^N - \left(\alpha_k^N\right)^2. \tag{C.23}$$

Proof Let us first prove Equation (C.22). Keeping in mind the density function defined by Equation (C.5) and exploiting the linearity of mean:

$$\begin{aligned} Cov[f_k(\mathbf{P}_N),\ f_l(\mathbf{P}_N)] &= Cov[f_l(\mathbf{P}_N),\ f_k(\mathbf{P}_N)] = \\ &= E\left[\left(f_k(\mathbf{P}_N) - \alpha_k^N\right) \cdot \left(f_l(\mathbf{P}_N) - \alpha_l^N\right)\right] = \\ = E[f_k(\mathbf{P}_N) \cdot f_l(\mathbf{P}_N)] - \alpha_l^N \cdot E[f_k(\mathbf{P}_N)] - \alpha_k^N \cdot E[f_l(\mathbf{P}_N)] + \alpha_k^N \cdot \alpha_l^N &= \\ &= E[f_k(\mathbf{P}_N) \cdot f_l(\mathbf{P}_N)] - \alpha_k^N \alpha_l^N. \end{aligned} \tag{C.24}$$

Now, noting that $Var[f_k(\mathbf{P}_N)] = Cov[f_k(\mathbf{P}_N),\ f_k(\mathbf{P}_N)])$ and $f_k(\mathbf{P}_N)^2 = f_k(\mathbf{P}_N)$, we have:

$$Var[f_k(\mathbf{P}_N)] = E\left[f_k(\mathbf{P}_N)^2\right] - \left(\alpha_k^N\right)^2 = \alpha_k^N - \left(\alpha_k^N\right)^2.$$

□

Lemma 57 *Consider* r. v. $f_k(\mathbf{P}_N)$ *and* $f_l(\mathbf{P}_N)$*, with $k, l \in \mathcal{N}$ and $k \neq l$ then the covariance matrix $\Sigma_{k;l} = (\sigma_{i,j})_{i,j \in \{k,l\}}$ with $\sigma_{i,j} = Cov\left[f_i(\mathbf{P}_N),\ f_j(\mathbf{P}_N)\right]$ has the following form:*

- *if $k \cap l = \emptyset$ then:*

$$\Sigma_{k;l} = \begin{pmatrix} \alpha_k^N - \left(\alpha_k^N\right)^2 & -\alpha_k^N \cdot \alpha_l^N \\ -\alpha_k^N \cdot \alpha_l^N & \alpha_l^N - \left(\alpha_l^N\right)^2 \end{pmatrix}, \tag{C.25}$$

- *if $k \subset l$ then:*

$$\Sigma_{k;l} = \begin{pmatrix} \alpha_k^N - \left(\alpha_k^N\right)^2 & \alpha_k^N \cdot \left(1 - \alpha_l^N\right) \\ \alpha_k^N \cdot \left(1 - \alpha_l^N\right) & \alpha_l^N - \left(\alpha_l^N\right)^2 \end{pmatrix}, \tag{C.26}$$

- *if* $k \cap l \neq \emptyset \wedge k \not\subset l \wedge l \not\subset k$ *then:*

$$\Sigma_{k;l} = \begin{pmatrix} \alpha_k^N - \left(\alpha_k^N\right)^2 & \alpha_{k\cap l}^N - \alpha_k^N \cdot \alpha_l^N \\ \alpha_{k\cap l}^N - \alpha_k^N \cdot \alpha_l^N & \alpha_l^N - \left(\alpha_l^N\right)^2 \end{pmatrix}, \tag{C.27}$$

Proof First, note that for any $k, l \in \mathcal{N}$ by Lemma 56:

$$Var\left[f_k\left(\mathbf{P}_N\right)\right] = \alpha_k^N - \left(\alpha_k^N\right)^2, \quad Var\left[f_l\left(\mathbf{P}_N\right)\right] = \alpha_l^N - \left(\alpha_l^N\right)^2.$$

Now let us provide the covariances for each of the cases under consideration, note that $k \neq l$:

- $k \cap l = \emptyset$**:** The condition yields that $f_k\left(\mathbf{P}_N\right) \cdot f_l\left(\mathbf{P}_N\right) = 0$ and thus by Lemma 56 we obtain:

$$Cov\left[f_k\left(\mathbf{P}_N\right),\ f_l\left(\mathbf{P}_N\right)\right] = Cov\left[f_l\left(\mathbf{P}_N\right),\ f_k\left(\mathbf{P}_N\right)\right] = -\alpha_k^N \alpha_l^N.$$

- $k \subset l$**:** The condition implies that $f_k\left(\mathbf{P}_N\right) \cdot f_l\left(\mathbf{P}_N\right) = f_k\left(\mathbf{P}_N\right)$ and thus by Lemma 56 we obtain:

$$Cov\left[f_k\left(\mathbf{P}_N\right),\ f_l\left(\mathbf{P}_N\right)\right] = Cov\left[f_l\left(\mathbf{P}_N\right),\ f_k\left(\mathbf{P}_N\right)\right] = \alpha_k^N - \alpha_k^N \cdot \alpha_l^N = \alpha_k^N \cdot \left(1 - \alpha_l^N\right).$$

- $k \cap l \neq \emptyset \wedge k \not\subset l \wedge l \not\subset k$**:** From the condition it follows that $f_k\left(\mathbf{P}_N\right) \cdot f_l\left(\mathbf{P}_N\right) = f_{k\cap l}\left(\mathbf{P}_N\right)$ and thus by Lemma 56 we obtain:

$$Cov\left[f_k\left(\mathbf{P}_N\right),\ f_l\left(\mathbf{P}_N\right)\right] = Cov\left[f_l\left(\mathbf{P}_N\right),\ f_k\left(\mathbf{P}_N\right)\right] = \alpha_{k\cap l}^N - \alpha_k^N \alpha_l^N.$$

□

The following lemma uses multi-dimensional Central Limit Theorem 55 in order to prove the dependency of the *r. v.* defining the *c. i.*, when $M \to \infty$.

Lemma 58 *For all* $k \in \mathcal{N}$, $\alpha_k^N \notin \{0, 1\}$, *the limit:*

$$\lim_{M\to\infty} \left(\frac{\sum_{i=1}^M f_k\left(\mathbf{P}_N^i\right) - M\alpha_k^N}{\sqrt{M}} \right) = L_k, \tag{C.28}$$

exists. The pairs of r. v. L_k *and* L_l *for any* $k, l \in \mathcal{N}$ *and* $k \neq l$ *are dependent.*

Proof The Central Limit Theorem 16 guarantees existence of the limit defined by Equation (C.28) in case $f_k\left(\mathbf{P}_N\right)$ has a non-zero variance. From Lemma 56 the variance of $f_k\left(\mathbf{P}_N\right)$ equals to $\alpha_k^N - \left(\alpha_k^N\right)^2 \neq 0$ for $\alpha_k^N \notin \{0, 1\}$.

It is known that if two *r. v.* are independent then their covariance is zero. Therefore in order to show the dependency of the *r. v.* L_g and L_t it suffices to show that $Cov\left[L_k,\ L_l\right] \neq 0$.

Since for any $k \in \mathcal{N}$ we have $E\left[\left(f_k\left(\mathbf{P}_N\right)\right)^2\right] < \infty$ then the following limit exists due to multi-dimensional Central Limit Theorem 55:

$$\lim_{M\to\infty}\left(\frac{\sum_{i=1}^{M}\left(f_g\left(\mathbf{P}_N^i\right), f_t\left(\mathbf{P}_N^i\right)\right) - M\left(\alpha_g^N, \alpha_t^N\right)}{\sqrt{M}}\right) = \left(L_g, L_t\right). \tag{C.29}$$

Note that the *r. v.* L_k and L_l obtained from Equation (C.28) are the same as obtained by Equation (C.29), because all operations on the vectors $\left(f_g\left(\mathbf{P}_N^i\right), f_t\left(\mathbf{P}_N^i\right)\right)$ are applied component wise. Also, thanks to multi-dimensional Central Limit Theorem 55, we know that the *r. v.* L_k and L_l have the covariance matrix $\Sigma_{k;l}$ defined by Lemma 57. The latter means that $Cov\left[L_k,\ L_l\right] = Cov\left[f_k\left(\mathbf{P}_N\right),\ f_l\left(\mathbf{P}_N\right)\right]$.

Since $|\mathcal{N}| = 5$ and we have to prove the pairwise dependency of the *r. v.*, consider the following ten cases:

1. **The *r. v.* L_g and L_t:** We have $\{g\} \cap \{t\} = \emptyset$ and thus by Lemma 57 it follows that:
$$Cov\left[L_g,\ L_t\right] = -\alpha_g^N \cdot \alpha_t^N \neq 0,$$
since for any $k \in \mathcal{N}$ we have $\alpha_k^N \notin \{0, 1\}$.

2. **The *r. v.* L_b and L_t:** Similar to the case 1, since $\{b\} \cap \{t\} = \emptyset$.

3. **The *r. v.* L_g and L_b:** Similar to the case 1, since $\{g\} \cap \{b\} = \emptyset$.

4. **The *r. v.* L_b and $L_{g,t}$:** Similar to the case 1, since $\{b\} \cap \{g, t\} = \emptyset$.

5. **The *r. v.* L_g and $L_{b,t}$:** Similar to the case 1, since $\{g\} \cap \{b, t\} = \emptyset$.

6. **The *r. v.* L_g and $L_{g,t}$:** Similar to the case 1. Notice that $\{g\} \subset \{g, t\}$ and thus:
$$Cov\left[L_g,\ L_{g,t}\right] = \alpha_g^N \cdot \left(1 - \alpha_{g,t}^N\right) \neq 0,$$
since for any $k \in \mathcal{N}$ we have $\alpha_k^N \notin \{0, 1\}$.

7. **The *r. v.* L_t and $L_{g,t}$:** Similar to the case 6, since $\{t\} \subset \{g, t\}$.

8. **The *r. v.* L_b and $L_{b,t}$:** Similar to the case 6, since $\{b\} \subset \{b, t\}$.

9. **The *r. v.* L_t and $L_{b,t}$:** Similar to the case 6, since $\{t\} \subset \{b, t\}$.

10. **The *r. v.* $L_{g,t}$ and $L_{b,t}$:** Similar to the case 1. Notice that $\{g, t\} \cap \{b, t\} \neq \emptyset$, $\{g, t\} \not\subset \{b, t\}$ and $\{b, t\} \not\subset \{g, t\}$ and thus:
$$\begin{aligned} Cov\left[L_{g,t},\ L_{b,t}\right] &= \alpha_t^N - \alpha_{g,t}^N \cdot \alpha_{b,t}^N = \text{ by Proposition 18 } = \\ &= \alpha_t^N - \left(\alpha_g^N + \alpha_t^N\right) \cdot \left(\alpha_b^N + \alpha_t^N\right) = \left(1 - \alpha_g^N - \alpha_b^N\right) \cdot \alpha_t^N - \alpha_g^N \cdot \alpha_b^N - \left(\alpha_t^N\right)^2 = \\ &= \text{ by Proposition 18 } = \left(\alpha_t^N\right)^2 - \alpha_g^N \cdot \alpha_b^N - \left(\alpha_t^N\right)^2 = -\alpha_g^N \cdot \alpha_b^N \neq 0, \end{aligned}$$
since for any $k \in \mathcal{N}$ we have $\alpha_k^N \notin \{0, 1\}$.

□

Proposition 59 *Let* $\overrightarrow{\mathbf{P}_N} = \left(\mathbf{P}_N^1, \ldots, \mathbf{P}_N^M\right)$ *be a sample of* $M \in \mathbb{N}_{\geq 2}$ *independent observations, and* $\alpha_k^N \notin \{0,1\}$ *for any* $k \in \mathcal{N}$. *The* c. i. *of* α_k^N *and* α_l^N *for any* $k, l \in \mathcal{N}$ *and* $k \neq l$, *derived using Equation (6.2) with* $\sigma_k^N = Var\left[f_k\left(\mathbf{P}_N^i\right)\right]$ *used in place of* $\overline{V}_k^N$, *are dependent in the limit of* $M \to \infty$.

Proof The *c. i.* mentioned above are based on the *r. v.* defined by the limit:

$$\lim_{M\to\infty}\left(\frac{\sum_{i=1}^{M} f_k\left(\mathbf{P}_N^i\right) - M\alpha_k^N}{\sigma_k^N \cdot \sqrt{M}}\right) = L'_k.$$

Lemma 58 proves dependency of the *r. v.* defined by the similar limit. The only difference is that the variance σ_k^N is excluded from the divider of the expression under the limit in Equation (58). This causes the *r. v.* L_k to converge to the normal distribution $N\left(0, \left(\sigma_k^N\right)^2\right)$, whereas L'_k converges to standard normal distribution $N(0,1)$. In fact, it is clear that the *r. v.* $L_k/\sigma_k^N = L'_k$, because:

$$\begin{aligned}\frac{1}{\sigma_k^N} \cdot L_k &= \frac{1}{\sigma_k^N} \cdot \lim_{M\to\infty}\left(\frac{\sum_{i=1}^{M} f_k\left(\mathbf{P}_N^i\right) - M\alpha_k^N}{\sqrt{M}}\right) = \\ &= \lim_{M\to\infty}\left(\frac{\sum_{i=1}^{M} f_k\left(\mathbf{P}_N^i\right) - M\alpha_k^N}{\sigma_k^N \cdot \sqrt{M}}\right) = L'_k\end{aligned}$$

The latter means that for any $k, l \in \mathcal{N}$: L_k and L_l are dependent **iff** L'_k and L'_l are dependent. This concludes the proof. □

C.1.2 Confidence intervals, the closed form

The following theorem gives us the possible *c. i.* of α_g, based on independent samples.

Theorem 60 *For independent samples* $\overrightarrow{\mathbf{P}_N}$, $\overrightarrow{\mathbf{P}'_N}$ *and* $\overrightarrow{\mathbf{P}''_N}$ *of* $M \in \mathbb{N}_{\geq 2}$ *independent observations each, and the* c. i. *of* α_k^N *for all* $k \in \mathcal{N}$ *with confidence* $1-\beta$, *the following*

holds:

$$Prob\left(A_l^g\left(\overrightarrow{\mathbf{P}_N}\right) \leq \alpha_g \leq A_r^g\left(\overrightarrow{\mathbf{P}_N}\right) + A_r^t\left(\overrightarrow{\mathbf{P}'_N}\right)\right) \succeq (1-\beta)\cdot\left(1-\frac{\beta}{2}\right), \quad \text{(C.30)}$$

$$Prob\left(1-\left(A_r^b\left(\overrightarrow{\mathbf{P}_N}\right) + A_r^t\left(\overrightarrow{\mathbf{P}'_N}\right)\right) \leq \alpha_g \leq 1 - A_l^b\left(\overrightarrow{\mathbf{P}_N}\right)\right) \succeq (1-\beta)\cdot\left(1-\frac{\beta}{2}\right), \quad \text{(C.31)}$$

$$Prob\left(A_l^g\left(\overrightarrow{\mathbf{P}_N}\right) \leq \alpha_g \leq A_r^{g,t}\left(\overrightarrow{\mathbf{P}'_N}\right)\right) \succeq \left(1-\frac{\beta}{2}\right)^2, \quad \text{(C.32)}$$

$$Prob\left(1 - A_r^{b,t}\left(\overrightarrow{\mathbf{P}_N}\right) \leq \alpha_g \leq 1 - A_l^b\left(\overrightarrow{\mathbf{P}'_N}\right)\right) \succeq \left(1-\frac{\beta}{2}\right)^2, \quad \text{(C.33)}$$

$$Prob\left(A_l^g\left(\overrightarrow{\mathbf{P}_N}\right) \leq \alpha_g \leq 1 - A_l^b\left(\overrightarrow{\mathbf{P}'_N}\right)\right) \succeq \left(1-\frac{\beta}{2}\right)^2, \quad \text{(C.34)}$$

$$Prob\left(1 - A_r^{b,t}\left(\overrightarrow{\mathbf{P}_N}\right) \leq \alpha_g \leq A_r^{g,t}\left(\overrightarrow{\mathbf{P}'_N}\right)\right) \succeq \left(1-\frac{\beta}{2}\right)^2, \quad \text{(C.35)}$$

$$Prob\left(1-\left(A_r^b\left(\overrightarrow{\mathbf{P}''_N}\right) + A_r^t\left(\overrightarrow{\mathbf{P}'_N}\right)\right) \leq \alpha_g \leq A_r^g\left(\overrightarrow{\mathbf{P}_N}\right) + A_r^t\left(\overrightarrow{\mathbf{P}'_N}\right)\right) \succeq \left(1-\frac{\beta}{2}\right)^3, \quad \text{(C.36)}$$

$$Prob\left(1-\left(A_r^b\left(\overrightarrow{\mathbf{P}''_N}\right) + A_r^t\left(\overrightarrow{\mathbf{P}_N}\right)\right) \leq \alpha_g \leq A_r^{g,t}\left(\overrightarrow{\mathbf{P}'_N}\right)\right) \succeq \left(1-\frac{\beta}{2}\right)^3, \quad \text{(C.37)}$$

$$Prob\left(1 - A_r^{b,t}\left(\overrightarrow{\mathbf{P}''_N}\right) \leq \alpha_g \leq A_r^g\left(\overrightarrow{\mathbf{P}_N}\right) + A_r^t\left(\overrightarrow{\mathbf{P}'_N}\right)\right) \succeq \left(1-\frac{\beta}{2}\right)^3. \quad \text{(C.38)}$$

Proof Consider Definition 20 for the sample $\overrightarrow{\mathbf{P}_N}$, then let us agree to add the prime and the double prime symbols to the events obtained from the samples $\overrightarrow{\mathbf{P}'_N}$ and $\overrightarrow{\mathbf{P}''_N}$ correspondingly. For example $E_r^{'t}$ and $E_r^{''g,t}$. Remember that events being obtained from independent samples are independent.

Let us derive the above mentioned confidence intervals, using Proposition 43 of Appendix C.1:

1. **Equation (C.30):** Using Proposition 43 with $A_l = \alpha_g^N$ and $A_r = \alpha_g^N + \alpha_t^N$ we get:

   $$E^g \wedge E_r^{'t} \Longrightarrow A_l^g\left(\overrightarrow{\mathbf{P}_N}\right) \leq \alpha_g \leq A_r^g\left(\overrightarrow{\mathbf{P}_N}\right) + A_r^t\left(\overrightarrow{\mathbf{P}'_N}\right),$$

   which, because the probability of the consequent event is always greater or equal than the probability of the antecedent event, implies:

   $$Prob\left(A_l^g\left(\overrightarrow{\mathbf{P}_N}\right) \leq \alpha_g \leq A_r^g\left(\overrightarrow{\mathbf{P}_N}\right) + A_r^t\left(\overrightarrow{\mathbf{P}'_N}\right)\right) \geq Prob\left(E^g \wedge E_r^{'t}\right). \quad \text{(C.39)}$$

   Using Equation (6.7) of Section 6.2.3 and noting that the events E^g and E_r^t are independent we have:

   $$Prob\left(E^g \wedge E_r^{'t}\right) = Prob\left(E^g\right)\cdot Prob\left(E_r^{'t}\right) \approx (1-\beta)\cdot\left(1-\frac{\beta}{2}\right),$$

   and therefore, by Definition 22, Equation (C.39) implies Equation (C.30).

   All the remaining cases are proved similarly.

2. **Equation (C.31):** Consider $A_l = 1 - \left(\alpha_b^N + \alpha_t^N\right)$, $A_r = 1 - \alpha_b^N$, E^b and $E_r^{'t}$.

3. **Equation (C.32):** Consider $A_l = \alpha_g^N$, $A_r = \alpha_{g,t}^N$, E_l^g and $E_r^{'g,t}$.
4. **Equation (C.33):** Consider $A_l = 1 - \alpha_{b,t}^N$, $A_r = 1 - \alpha_b^N$, $E_r^{b,t}$ and $E_l^{'b}$.
5. **Equation (C.34):** Consider $A_l = \alpha_g^N$, $A_r = 1 - \alpha_b^N$, E_l^g and $E_l^{'b}$.
6. **Equation (C.35):** Consider $A_l = 1 - \alpha_{b,t}^N$, $A_r = \alpha_{g,t}^N$, $E_r^{b,t}$ and $E_r^{'g,t}$.
7. **Equation (C.36):** Consider $A_l = 1 - \left(\alpha_b^N + \alpha_t^N\right)$, $A_r = \alpha_g^N + \alpha_t^N$, $E_r^{'t}$, $E_r^{''b}$ and E_r^g.
8. **Equation (C.37):** Consider $A_l = 1 - \left(\alpha_b^N + \alpha_t^N\right)$, $A_r = \alpha_{g,t}^N$, E_r^t, $E_r^{''b}$ and $E_r^{'g,t}$.
9. **Equation (C.38):** Consider $A_l = 1 - \alpha_{b,t}^N$, $A_r = \alpha_g^N + \alpha_t^N$, $E_r^{'t}$, $E_r^{''b,t}$ and E_r^g.

□

Proposition 61 *For any $N \in \mathbb{N}_{\geq 0}$, two independent samples $\overrightarrow{\mathbf{P}_N}$ and $\overrightarrow{\mathbf{P}'_N}$ of $M \in \mathbb{N}_{>0}$ independent observations each, the following limit holds* a. s.

$$\lim_{M\to\infty}\left(\left|\frac{\Gamma^k\left(\overrightarrow{\mathbf{P}_N}\right)}{M} - \frac{\Gamma^k\left(\overrightarrow{\mathbf{P}'_N}\right)}{M}\right|\right) = 0 \tag{C.40}$$

Proof Applying the strong law of large numbers for Bernoulli trials discussed in Section 5.7 we obtain:

$$\mathit{Prob}\left(\lim_{M\to\infty}\left(\frac{\Gamma^k\left(\overrightarrow{\mathbf{P}_N}\right)}{M}\right) = \alpha_k^N\right) = 1, \quad \mathit{Prob}\left(\lim_{M\to\infty}\left(\frac{\Gamma^k\left(\overrightarrow{\mathbf{P}'_N}\right)}{M}\right) = \alpha_k^N\right) = 1$$

Due to $\overrightarrow{\mathbf{P}_N}$ and $\overrightarrow{\mathbf{P}'_N}$ being independent we have:

$$\mathit{Prob}\left(\lim_{M\to\infty}\left(\frac{\Gamma^k\left(\overrightarrow{\mathbf{P}_N}\right)}{M}\right) = \alpha_k^N \quad \bigwedge \quad \lim_{M\to\infty}\left(\frac{\Gamma^k\left(\overrightarrow{\mathbf{P}'_N}\right)}{M}\right) = \alpha_k^N\right) = 1, \tag{C.41}$$

and now it suffices to notice the implication:

$$\lim_{M\to\infty}\left(\frac{\Gamma^k\left(\overrightarrow{\mathbf{P}_N}\right)}{M}\right) = \alpha_k^N \quad \bigwedge \quad \lim_{M\to\infty}\left(\frac{\Gamma^k\left(\overrightarrow{\mathbf{P}'_N}\right)}{M}\right) = \alpha_k^N \Longrightarrow$$
$$\Longrightarrow \lim_{M\to\infty}\left(\left|\frac{\Gamma^k\left(\overrightarrow{\mathbf{P}_N}\right)}{M} - \frac{\Gamma^k\left(\overrightarrow{\mathbf{P}'_N}\right)}{M}\right|\right) = 0,$$

which with Equation (C.41) gives that Equation (C.40) holds *a. s.* □

Lemma 62 *For any fixed confidence $1-\beta$, $k \in \mathcal{N}$ and $M \in \mathbb{N}_{\geq 2}$ the following holds:*

$$\sqrt{\frac{\Gamma_N^k \cdot \left(M - \Gamma_N^k\right)}{M \cdot (M-1)}} \leq \sqrt{2}$$

Proof First of all notice that $\Gamma_N^k \in \mathbb{N}_{[0,M]}$ and thus:

$$0 \leq \frac{\Gamma_N^k \cdot \left(M - \Gamma_N^k\right)}{M \cdot (M-1)} \leq \frac{M - \Gamma_N^k}{M-1}.$$

The right-hand side of the inequality is a linear function of Γ_N^k which monotonously decreases with increase of Γ_N^k and therefore we have:

$$\frac{M - \Gamma_N^k}{M-1} \leq \frac{M}{M-1}.$$

To conclude the proof we should show that the right hand side of the last inequality is less or equal than 2.

Consider the function $\frac{x}{x-1}$ with $x \in \mathbb{R}_{\geq 2}$. Notice that for $x = 2$ we have $\frac{x}{x-1} = 2$ and the function is decreasing on $\mathbb{R}_{\geq 2}$ since there its first derivative is negative:

$$\left(\frac{x}{x-1}\right)' = -\frac{1}{(x-1)^2} < 0.$$

Therefore we conclude that $\frac{x}{x-1} \leq 2$ on $\mathbb{R}_{\geq 2}$, implying that $\frac{M}{M-1} \leq 2$ on $\mathbb{N}_{\geq 2}$. □

Lemma 63 *For a fixed confidence* $1-\beta$, $k \in \mathcal{N}$ *and* $M \in \mathbb{N}_{\geq 2}$ *the following holds:*

$$\overline{X}_k^N - \sqrt{2} \cdot \frac{\widetilde{z_n}(\beta)}{\sqrt{M}} \leq A_l^k\left(\Gamma_N^k\right) \leq \overline{X}_k^N \leq A_r^k\left(\Gamma_N^k\right) \leq \overline{X}_k^N + \sqrt{2} \cdot \frac{\widetilde{z_n}(\beta)}{\sqrt{M}}.$$

Proof Follows directly from Lemma 46 of Appendix C.1, Lemma 62 above, and Equations (6.4). □

Theorem 64 *For any* $N \in \mathbb{N}_{\geq 0}$, *confidence* $1-\beta$, *two independent samples* $\overrightarrow{\mathbf{P}_N}$ *and* $\overrightarrow{\mathbf{P}'_N}$ *of* $M \in \mathbb{N}_{>0}$ *independent observations each, the following holds:*

$$Prob\left(\lim_{M\to\infty}\left(\left|A_l^k\left(\overrightarrow{\mathbf{P}_N}\right) - A_l^k\left(\overrightarrow{\mathbf{P}'_N}\right)\right|\right) = 0\right) = 1, \tag{C.42}$$

$$Prob\left(\lim_{M\to\infty}\left(\left|A_r^k\left(\overrightarrow{\mathbf{P}_N}\right) - A_r^k\left(\overrightarrow{\mathbf{P}'_N}\right)\right|\right) = 0\right) = 1. \tag{C.43}$$

Proof Clearly, it suffices to prove one of Equations (C.42) and (C.43) due to the symmetric structure of $A_l^k(.)$ and $A_r^k(.)$. Let us choose Equation (C.42) and analyze the bounds of $\left|A_l^k\left(\overrightarrow{\mathbf{P}_N}\right) - A_l^k\left(\overrightarrow{\mathbf{P}'_N}\right)\right|$ in the deterministic sense. From Lemma 63 it follows that:

$$\frac{\Gamma^k\left(\overrightarrow{\mathbf{P}_N}\right)}{M} - \sqrt{2} \cdot \frac{\widetilde{z_n}(\beta)}{\sqrt{M}} \leq A_l^k\left(\overrightarrow{\mathbf{P}_N}\right) \leq \frac{\Gamma^k\left(\overrightarrow{\mathbf{P}_N}\right)}{M},$$

$$\frac{\Gamma^k\left(\overrightarrow{\mathbf{P}'_N}\right)}{M} - \sqrt{2} \cdot \frac{\widetilde{z_n}(\beta)}{\sqrt{M}} \leq A_l^k\left(\overrightarrow{\mathbf{P}'_N}\right) \leq \frac{\Gamma^k\left(\overrightarrow{\mathbf{P}'_N}\right)}{M},$$

and therefore, for $D_M^k = \frac{\Gamma^k\left(\overrightarrow{\mathbf{P}_N}\right)}{M} - \frac{\Gamma^k\left(\overrightarrow{\mathbf{P}'_N}\right)}{M}$ we have:

$$D_M^k - \sqrt{2}\cdot\frac{\widetilde{z_n}(\beta)}{\sqrt{M}} \leq A_l^k\left(\overrightarrow{\mathbf{P}_N}\right) - A_l^k\left(\overrightarrow{\mathbf{P}'_N}\right) \leq D_M^k + \sqrt{2}\cdot\frac{\widetilde{z_n}(\beta)}{\sqrt{M}}. \tag{C.44}$$

Obviously, $-\left|D_M^k\right| \leq D_M^k \leq \left|D_M^k\right|$ and thus Equation (C.44) can be transformed into:

$$-\left|D_M^k\right| - \sqrt{2}\cdot\frac{\widetilde{z_n}(\beta)}{\sqrt{M}} \leq A_l^k\left(\overrightarrow{\mathbf{P}_N}\right) - A_l^k\left(\overrightarrow{\mathbf{P}'_N}\right) \leq \left|D_M^k\right| + \sqrt{2}\cdot\frac{\widetilde{z_n}(\beta)}{\sqrt{M}},$$

that in its turn, due to $\widetilde{z_n}(\beta) \geq 0$, is equivalent to:

$$\left|A_l^k\left(\overrightarrow{\mathbf{P}_N}\right) - A_l^k\left(\overrightarrow{\mathbf{P}'_N}\right)\right| \leq \left|D_M^k\right| + \sqrt{2}\cdot\frac{\widetilde{z_n}(\beta)}{\sqrt{M}}. \tag{C.45}$$

Without a doubt $\lim\limits_{M\to\infty}\left(\sqrt{2}\cdot\frac{\widetilde{z_n}(\beta)}{\sqrt{M}}\right) = 0$ and by Proposition 61 we have:

$$Prob\left(\lim_{M\to\infty}(|D_M|) = 0\right) = 1. \tag{C.46}$$

Due to Equation (C.45) the following implication holds:

$$\begin{aligned}\left(\lim_{M\to\infty}(|D_M|) = 0 \bigwedge \lim_{M\to\infty}\left(\sqrt{2}\cdot\frac{\widetilde{z_n}(\beta)}{\sqrt{M}}\right) = 0\right) \Longrightarrow \\ \Longrightarrow \lim_{M\to\infty}\left(\left|A_l^k\left(\overrightarrow{\mathbf{P}_N}\right) - A_l^k\left(\overrightarrow{\mathbf{P}'_N}\right)\right|\right) = 0,\end{aligned}$$

from which, by Equation (C.46), we conclude that Equation (C.42) holds too. □

Lemma 65 *For a fixed confidence* $1-\beta$ *and a sample* $\overrightarrow{\mathbf{P}_N}$ *of* $M \in \mathbb{N}_{\geq 2}$ *observations:*

$$A_l^g\left(\overrightarrow{\mathbf{P}_N}\right) = 1 - A_r^{b,t}\left(\overrightarrow{\mathbf{P}_N}\right), \quad A_r^{g,t}\left(\overrightarrow{\mathbf{P}_N}\right) = 1 - A_l^b\left(\overrightarrow{\mathbf{P}_N}\right).$$

Proof First, using Lemma 45 of Appendix C.1 let us notice that:

$$\begin{aligned}A_l^g\left(\overrightarrow{\mathbf{P}_N}\right) &= \overline{X}_g^N - \frac{\widetilde{z_n}(\beta)}{M}\cdot\overline{V}_g^N, \quad 1 - A_r^{b,t}\left(\overrightarrow{\mathbf{P}_N}\right) = \overline{X}_g^N - \frac{\widetilde{z_n}(\beta)}{M}\cdot\overline{V}_{b,t}^N \\ A_r^{g,t}\left(\overrightarrow{\mathbf{P}_N}\right) &= \overline{X}_{g,t}^N + \frac{\widetilde{z_n}(\beta)}{M}\cdot\overline{V}_{g,t}^N, \quad 1 - A_l^b\left(\overrightarrow{\mathbf{P}_N}\right) = \overline{X}_{g,t}^N + \frac{\widetilde{z_n}(\beta)}{M}\cdot\overline{V}_b^N\end{aligned}$$

Therefore, in order to prove the claim of this lemma we have to show that:

$$\overline{V}_g^N = \overline{V}_{b,t}^N, \quad \text{and} \quad \overline{V}_{g,t}^N = \overline{V}_b^N. \tag{C.47}$$

Second, consider the fact that by Lemma 44 of Appendix C.1 we have:

$$\Gamma_N^g = M - \Gamma_N^{b,t}, \quad \text{and} \quad \Gamma_N^{g,t} = M - \Gamma_N^b.$$

Then using the representation of $\overline{V}_k^N$ given by Equation (6.5) it is clear that Equations (C.47) hold. □

Lemma 66 *For a fixed confidence* $1-\beta$, $M \in \mathbb{N}_{\geq 2}$, $N \in \mathbb{N}_{\geq 0}$ *and sample* $\overrightarrow{\mathbf{P}_N}$ *of* M *observations the following holds:*

$$A_r^{g,t}\left(\overrightarrow{\mathbf{P}_N}\right) \leq A_r^{g}\left(\overrightarrow{\mathbf{P}_N}\right) + A_r^{t}\left(\overrightarrow{\mathbf{P}_N}\right), \quad A_r^{b,t}\left(\overrightarrow{\mathbf{P}_N}\right) \leq A_r^{b}\left(\overrightarrow{\mathbf{P}_N}\right) + A_r^{t}\left(\overrightarrow{\mathbf{P}_N}\right). \tag{C.48}$$

Proof Clearly, it suffices to prove one of the given inequalities, since the proof for the other one is equivalent.

Using Equation (6.4), the first inequality of Equation (C.48) can be rewritten as:

$$\overline{X}_{g,t}^N + \frac{\widetilde{z_n}(\beta)\cdot \overline{V}_{g,t}^N}{\sqrt{M}} \leq \overline{X}_{g}^N + \frac{\widetilde{z_n}(\beta)\cdot \overline{V}_{g}^N}{\sqrt{M}} + \overline{X}_{t}^N + \frac{\widetilde{z_n}(\beta)\cdot \overline{V}_{t}^N}{\sqrt{M}}. \tag{C.49}$$

Here $\widetilde{z_n}(\beta) \geq 0$, and $\overline{X}_{g,t}^N = \overline{X}_{g}^N + \overline{X}_{t}^N$ by Lemma 45. Therefore, in case $\widetilde{z_n}(\beta) = 0$, Equation (C.49) trivially holds, and for $\widetilde{z_n}(\beta) > 0$ it is equivalent to:

$$\overline{V}_{g,t}^N \leq \overline{V}_{g}^N + \overline{V}_{t}^N,$$

that using Equation (6.5) can be rewritten as:

$$\sqrt{\sum_{i=1}^{M}\left(f_{g,t}\left(\mathbf{P}_N^i\right) - \frac{\Gamma_N^{g,t}}{M}\right)^2} \leq \sqrt{\sum_{i=1}^{M}\left(f_{g}\left(\mathbf{P}_N^i\right) - \frac{\Gamma_N^{g}}{M}\right)^2} + \sqrt{\sum_{i=1}^{M}\left(f_{t}\left(\mathbf{P}_N^i\right) - \frac{\Gamma_N^{t}}{M}\right)^2}.$$

Considering the fact that $f_{g,t}\left(\mathbf{P}_N^i\right) = f_g\left(\mathbf{P}_N^i\right) + f_t\left(\mathbf{P}_N^i\right)$ for any $i \in \mathbb{N}_{[1,M]}$ (and thus $\Gamma_N^{g,t} = \Gamma_N^g + \Gamma_N^t$), the latter is equivalent to:

$$\sqrt{\sum_{i=1}^{M}\left(\gamma_i + \delta_i\right)^2} \leq \sqrt{\sum_{i=1}^{M}\gamma_i^2} + \sqrt{\sum_{i=1}^{M}\delta_i^2}, \tag{C.50}$$

with $\gamma_i = f_g\left(\mathbf{P}_N^i\right) - \frac{\Gamma_N^g}{M}$ and $\delta_i = f_t\left(\mathbf{P}_N^i\right) - \frac{\Gamma_N^t}{M}$.

Equation (C.50) holds, because it is the triangle inequality for the M-dimensional Euclidean space. □

Lemma 67 *For a fixed confidence* $1-\beta$, $N \in \mathbb{N}_{\geq 0}$, *and a finite-state DTMC* $\mathbf{P}$, *with a positive probability there exist independent samples* $\overrightarrow{\mathbf{P}_N}$, $\overrightarrow{\mathbf{P}'_N}$ *and* $\overrightarrow{\mathbf{P}''_N}$ *of* $M \in \mathbb{N}_{\geq 2}$ *independent observations each, such that:*

$$A_r^{g,t}\left(\overrightarrow{\mathbf{P}_N}\right) \leq A_r^{g}\left(\overrightarrow{\mathbf{P}''_N}\right) + A_r^{t}\left(\overrightarrow{\mathbf{P}'_N}\right), \quad \textit{and} \quad A_r^{b,t}\left(\overrightarrow{\mathbf{P}_N}\right) \leq A_r^{b}\left(\overrightarrow{\mathbf{P}''_N}\right) + A_r^{t}\left(\overrightarrow{\mathbf{P}'_N}\right). \tag{C.51}$$

Proof Clearly, it suffices to prove one of the given inequalities, since the proof for the other one is equivalent.

Let $\mathbf{P}_N$ be a *r. v.* defining the state of $\mathbf{P}$ at epoch N. Then, since we have a finite-state Markov chain and $N < \infty$, a sample $\overrightarrow{\mathbf{P}_N}$ obtained via simulations of $\mathbf{P}_N$ has a positive probability to appear. The latter means that, continuing simulations, with a positive probability we can obtain an independent samples $\overrightarrow{\mathbf{P}_N}$, $\overrightarrow{\mathbf{P}'_N}$ and $\overrightarrow{\mathbf{P}''_N}$ that are equal up to the permutation of observations.

Now, to conclude the proof, it suffices to show that the first of Equations (C.51) holds for any samples $\overrightarrow{\mathbf{P}_N}$, $\overrightarrow{\mathbf{P}'_N}$ and $\overrightarrow{\mathbf{P}''_N}$ equal up to permutation of the observations. In the latter case, due to $\Gamma^{g,t}_N$, Γ^{g}_N and Γ^{t}_N having the same values on $\overrightarrow{\mathbf{P}_N}$, $\overrightarrow{\mathbf{P}'_N}$ and $\overrightarrow{\mathbf{P}''_N}$, it is enough to consider just one set of such samples. Let us take $\overrightarrow{\mathbf{P}_N} = \overrightarrow{\mathbf{P}'_N} = \overrightarrow{\mathbf{P}''_N}$ then the first of Equations (C.51) holds due to Lemma 66. □

Lemma 68 *For two* c. i. $\left[A^1_l, A^1_r\right]$ *and* $\left[A^2_l, A^2_r\right]$ *such that* $A^2_l \le A^1_r$ *and* $A^1_l \le A^2_r$ *Algorithm 3 gives a non-contradictory answer.*

Proof Let us consider only the case of $\bowtie \in \{\le\}$ because all the other cases are similar. Depending on the value of $b \in \mathbb{R}$, Algorithm 3 will give answers:

- For the *c. i.* $\left[A^1_l, A^1_r\right]$: *FALSE* if $b \in F = \left(-\infty, A^1_l\right)$, *NN* if $b \in N = \left[A^1_l, A^1_r\right)$, and *TRUE* if $b \in T = \left[A^1_r, +\infty\right)$
- For the *c. i.* $\left[A^2_l, A^2_r\right]$: *FALSE* if $b \in F' = \left(-\infty, A^2_l\right)$, *NN* if $b \in N' = \left[A^2_l, A^2_r\right)$, and *TRUE* if $b \in T' = \left[A^2_r, +\infty\right)$

In order to have *TRUE* and *FALSE* answers for the two *c. i.* simultaneously we need to have either $F \cup T' \neq \emptyset$ or $F' \cup T \neq \emptyset$. Which is impossible due to $A^2_l \le A^1_r$ and $A^1_l \le A^2_r$. □

C.1.3 The dependency from sample size and simulation length

Lemma 69 *For* $A^k_l\left(\overrightarrow{\mathbf{P}_N}\right)$ *and* $A^k_r\left(\overrightarrow{\mathbf{P}_N}\right)$ *given by Equation (6.4) let* $k \in \mathcal{N}$, $\widetilde{z_n}(\beta) \ge 0$, $M \in \mathbb{N}_{\ge 2}$ *and* $\Gamma^k_N \in \mathbb{N}_{[0,M]}$ *then the following holds:*

- $A^k_l\left(\Gamma^k_N\right) = 0$ **iff** $\Gamma^k_N = \frac{(\widetilde{z_n}(\beta))^2 \cdot M}{(\widetilde{z_n}(\beta))^2 + M - 1}$ *or* $\Gamma^k_N = 0$.
- $A^k_r\left(\Gamma^k_N\right) = 1$ **iff** $\Gamma^k_N = \frac{M \cdot (M-1)}{(\widetilde{z_n}(\beta))^2 + M - 1}$ *or* $\Gamma^k_N = M$.

Proof Below, without a loss of generality, we are going to treat $A^k_l\left(\Gamma^k_N\right)$ and $A^k_r\left(\Gamma^k_N\right)$ as functions of some variable $x \in \mathbb{R}_{[0,M]}$, i.e. we are going to solve two equation:

$$A^k_l(x) = 0, \quad \text{and} \quad A^k_r(x) = 1.$$

- From Equation (6.4) and Lemma 46 we have that $A^k_l(x) = 0$ is equivalent to:

$$\frac{x}{M} - \frac{\widetilde{z_n}(\beta)}{\sqrt{M}} \cdot \sqrt{\frac{x \cdot (M-x)}{M \cdot (M-1)}} = 0.$$

Which via a series of trivial transformations can be rewritten as:

$$\left(\left(\widetilde{z_n}\left(\beta\right)\right)^2 + M - 1\right) \cdot x^2 - M \cdot \left(\widetilde{z_n}\left(\beta\right)\right)^2 \cdot x = 0.$$

The latter has only two real roots, namely:

$$x_1 = \frac{\left(\widetilde{z_n}\left(\beta\right)\right)^2 \cdot M}{\left(\widetilde{z_n}\left(\beta\right)\right)^2 + M - 1}, \quad \text{and} \quad x_2 = 0.$$

Note that $0 \leq x_1 \leq M$, where the left part of the inequality is trivial and the right part is due to the fact that $\frac{\left(\widetilde{z_n}(\beta)\right)^2}{\left(\widetilde{z_n}(\beta)\right)^2 + M - 1} \leq 1$, since $M - 1 \geq 1$.

- From Equation (6.4) and Lemma 46 we have that $A_r^k\left(x\right) = 1$ is equivalent to:

$$\frac{x}{M} + \frac{\widetilde{z_n}\left(\beta\right)}{\sqrt{M}} \cdot \sqrt{\frac{x \cdot (M - x)}{M \cdot (M - 1)}} = 1.$$

Which via a series of trivial transformations can be rewritten as:

$$\left(\left(\widetilde{z_n}\left(\beta\right)\right)^2 + M - 1\right) \cdot x^2 - M \cdot \left(\left(\widetilde{z_n}\left(\beta\right)\right)^2 + 2 \cdot (M - 1)\right) \cdot x + M^2 \cdot (M - 1) = 0.$$

The latter has only two real roots, namely:

$$x_1 = \frac{M \cdot (M - 1)}{\left(\widetilde{z_n}\left(\beta\right)\right)^2 + M - 1}, \quad \text{and} \quad x_2 = M.$$

Note that $0 \leq x_1 \leq M$, where the left part of the inequality is trivial and the right part is due to the fact that $\frac{M-1}{\left(\widetilde{z_n}(\beta)\right)^2 + M - 1} \leq 1$, since $\left(\widetilde{z_n}\left(\beta\right)\right)^2 \geq 0$.

□

Lemma 70 *For $A_l^k\left(\overrightarrow{\mathbf{P}_N}\right)$ and $A_r^k\left(\overrightarrow{\mathbf{P}_N}\right)$ given by Equation (6.4) let $k \in \mathcal{N}$, $\widetilde{z_n}\left(\beta\right) \geq 0$, $M \in \mathbb{N}_{\geq 2}$ and $\Gamma_N^k \in \mathbb{N}_{(0,M)}$ then the following holds:*

- *$A_l^k\left(\Gamma_N^k\right)$ is increasing on $\mathbb{N}_{(x_1,M)}$, with $x_1 = \frac{M}{2} \cdot \left(1 - \sqrt{\frac{M-1}{\left(\widetilde{z_n}(\beta)\right)^2 + M - 1}}\right)$.*
- *$A_r^k\left(\Gamma_N^k\right)$ is increasing on $\mathbb{N}_{(0,x_2)}$, with $x_2 = \frac{M}{2} \cdot \left(1 + \sqrt{\frac{M-1}{\left(\widetilde{z_n}(\beta)\right)^2 + M - 1}}\right)$.*

Proof Below, without a loss of generality, we are going to treat $A_l^k\left(\Gamma_N^k\right)$ and $A_r^k\left(\Gamma_N^k\right)$ as functions of some variable $x \in \mathbb{R}_{[0,M]}$, i.e $A_l^k\left(x\right)$ and $A_r^k\left(x\right)$.

In order to prove that a function is increasing on some interval we shall show that its first derivative is strictly positive on it. For doing so we should find where the derivative turns into zero or does not exist and analyze its sign in between these points.

- The first derivative of $A_l^k\left(x\right)$ equals to:

$$\frac{1}{M} \cdot \left(1 - \frac{\widetilde{z_n}\left(\beta\right)}{\sqrt{M - 1}} \cdot \frac{M - 2 \cdot x}{2 \cdot \sqrt{x \cdot (M - x)}}\right). \tag{C.52}$$

Clearly it does not exist (turns into infinity) only when $x = 0$ or $x = M$. In order to find the points where it turns into zero, after a trivial rewriting of the derivative, we have to solve the following equation:

$$2 \cdot \sqrt{(M-1) \cdot x \cdot (M-x)} = \widetilde{z_n}(\beta) \cdot (M - 2 \cdot x). \tag{C.53}$$

Notice that the left-hand side of the equation is non-negative and thus the right hand side should be non-negative too, implying the condition $x \leq \frac{M}{2}$. Under this condition we can square both sides of the equation and after a series of trivial transformations obtain the following:

$$4 \cdot \left(\left(\widetilde{z_n}(\beta)\right)^2 + M - 1 \right) \cdot x^2 - 4 \cdot M \cdot \left(\left(\widetilde{z_n}(\beta)\right)^2 + M - 1 \right) \cdot x + M^2 \cdot \left(\widetilde{z_n}(\beta)\right)^2 = 0.$$

This equation has two real roots:

$$x_{1,2} = \frac{M}{2} \cdot \left(1 \pm \sqrt{\frac{M-1}{\left(\widetilde{z_n}(\beta)\right)^2 + M - 1}} \right)$$

Remember, that we should have $x \leq \frac{M}{2}$ and thus there is only one suitable root:

$$x_1 = \frac{M}{2} \cdot \left(1 - \sqrt{\frac{M-1}{\left(\widetilde{z_n}(\beta)\right)^2 + M - 1}} \right)$$

Now, on $\mathbb{N}_{(0,M)}$ the derivative of $A_l^k(x)$ exists and turns into zero only in x_1. Let us take $x = \frac{M}{2}$ that belongs to $\mathbb{N}_{(x_1,M)}$, then Equation (C.52) becomes equal to $\frac{1}{M}$. The latter is positive, meaning that the derivative of $A_l^k(x)$ is positive on $\mathbb{N}_{(x_1,M)}$ and thus $A_l^k(x)$ is increasing on it.

- The first derivative of $A_r^k(x)$ equals to:

$$\frac{1}{M} \cdot \left(1 + \frac{\widetilde{z_n}(\beta)}{\sqrt{M-1}} \cdot \frac{M - 2 \cdot x}{2 \cdot \sqrt{x \cdot (M-x)}} \right).$$

The rest of the proof is similar to the previous case, the only difference is that for zero points we get equation:

$$2 \cdot \sqrt{(M-1) \cdot x \cdot (M-x)} = -\widetilde{z_n}(\beta) \cdot (M - 2 \cdot x).$$

That has the same roots as Equation (C.53) but has to be solved under the condition $x \geq \frac{M}{2}$ and thus we shall take the root x_2.

□

Proposition 71 *For* $A_l^k\left(\overrightarrow{\mathbf{P}_N}\right)$ *and* $A_r^k\left(\overrightarrow{\mathbf{P}_N}\right)$ *given by Equation (6.4) let* $k \in \mathcal{N}$, $\widetilde{z_n}(\beta) \geq 0$, $M \in \mathbb{N}_{\geq 2}$, $\Gamma_N^k \in \mathbb{N}_{(0,M)}$ *then:*

$$\Gamma_1 = \frac{\left(\widetilde{z_n}(\beta)\right)^2 \cdot M}{\left(\widetilde{z_n}(\beta)\right)^2 + M - 1}, \quad \textit{and} \quad \Gamma_2 = \frac{M \cdot (M-1)}{\left(\widetilde{z_n}(\beta)\right)^2 + M - 1}.$$

- $A_l^k\left(\Gamma_N^k\right)$ *is increasing from* 0 *to* 1.0 *with the increase of* $\Gamma_N^k \in \mathbb{N}_{\left(\Gamma_1, M\right)}$,
- $A_r^k\left(\Gamma_N^k\right)$ *is increasing from* 0 *to* 1.0 *with the increase of* $\Gamma_N^k \in \mathbb{N}_{\left(0, \Gamma_2\right)}$.

Proof This proposition is an immediate consequence of Lemma 69 and Lemma 70 if we can show that:

$$\frac{M}{2} \cdot \left(1 - \sqrt{\frac{M-1}{\left(\widetilde{z_n}\left(\beta\right)\right)^2 + M - 1}}\right) \leq \Gamma_1, \tag{C.54}$$

$$\Gamma_2 \leq \frac{M}{2} \cdot \left(1 + \sqrt{\frac{M-1}{\left(\widetilde{z_n}\left(\beta\right)\right)^2 + M - 1}}\right). \tag{C.55}$$

First let us define $y = \sqrt{\frac{M-1}{\left(\widetilde{z_n}\left(\beta\right)\right)^2 + M - 1}}$, clearly $y \in \mathbb{R}_{(0,1]}$, and also notice that:

$$\Gamma_1 = M - \Gamma_2.$$

Then Equations (C.54) and (C.55) can be both rewritten as:

$$2 \cdot y^2 - y - 1 \leq 0, \tag{C.56}$$

that trivially holds since the function on the left-hand side turns into zero only at points $y_1 = -\frac{1}{2}$, $y_2 = 1$ and in between them its values are negative. Now remember that in our case $y \in \mathbb{R}_{(0,1]}$ and therefore Equations (C.54) and (C.55) hold. □

C.2 Steady-state operator

Theorem 72 *For the* c. i. *given by Equations (6.26) and (6.27), and based on independently obtained samples, the following* c. i. *results:*

$$Prob\left(\sum_{i=1}^{K} A_l^{s_0}\left(\overrightarrow{\mathbf{P}_i}\right) \cdot A_l^i \leq Prob^{\infty}\left(s_0,\, \mathcal{G}\right) \leq \sum_{i=1}^{K} A_r^{s_0}\left(\overrightarrow{\mathbf{P}_i'}\right) \cdot A_r^i\right) \succeq \prod_{i=1}^{K} \xi_i^r \xi_i^s, \tag{C.57}$$

under the assumption that for all $i \in I : A_l^{s_0}\left(\overrightarrow{\mathbf{P}_i}\right), A_r^{s_0}\left(\overrightarrow{\mathbf{P}_i'}\right), A_l^i, A_r^i \in \mathbb{R}_{[0,1]}$.

Proof First let us notice that $I = \{1, \ldots, K\}$. Then using the fact that the *c. i.* given by Equations (6.26) and (6.27) are based on independently obtained samples we can conclude that:

$$Prob\left(\bigwedge_{i \in I}\left(\left(A_l^i \leq \pi_i^g \leq A_r^i\right) \wedge \left(A_l^{s_0}\left(\overrightarrow{\mathbf{P}_i}\right) \leq p_i^{s_0} \leq A_r^{s_0}\left(\overrightarrow{\mathbf{P}_i'}\right)\right)\right)\right) \succeq \prod_{i=1}^{K} \xi_i^r \xi_i^s. \tag{C.58}$$

Keeping in mind that for all $i \in I : A_l^{s_0}\left(\overrightarrow{\mathbf{P}_i}\right), A_r^{s_0}\left(\overrightarrow{\mathbf{P}_i'}\right), A_l^i, A_r^i \in \mathbb{R}_{[0,1]}$ we have the following implication:

$$\begin{aligned}\left(\left(A_l^i \leq \pi_i^g \leq A_r^i\right) \wedge \left(A_l^{s_0}\left(\overrightarrow{\mathbf{P}_i}\right) \leq p_i^{s_0} \leq A_r^{s_0}\left(\overrightarrow{\mathbf{P}_i'}\right)\right)\right) \Longrightarrow \\ \Longrightarrow \left(A_l^{s_0}\left(\overrightarrow{\mathbf{P}_i}\right) \cdot A_l^i \leq p_i^{s_0} \cdot \pi_i^g \leq A_r^{s_0}\left(\overrightarrow{\mathbf{P}_i'}\right) \cdot A_r^i\right).\end{aligned} \tag{C.59}$$

Now it suffices to notice that:

$$\left(\bigwedge_{i\in I}\left(A_l^{s_0}\left(\overrightarrow{\mathbf{P}_i}\right)\cdot A_l^i \leq p_i^{s_0}\cdot\pi_i^g \leq A_r^{s_0}\left(\overrightarrow{\mathbf{P}_i'}\right)\cdot A_r^i\right)\right)\Longrightarrow$$
$$\Longrightarrow\left(\sum_{i=1}^{K}A_l^{s_0}\left(\overrightarrow{\mathbf{P}_i}\right)\cdot A_l^i \leq \sum_{i=1}^{K}p_i^{s_0}\cdot\pi_i^g \leq \sum_{i=1}^{K}A_r^{s_0}\left(\overrightarrow{\mathbf{P}_i'}\right)\cdot A_r^i\right), \tag{C.60}$$

where the right-hand side gives the *c. i.* borders for $Prob^{\infty}\left(s_0,\ \mathcal{G}\right)$, see Equation (6.25), that is obtained by summing up all the inequalities from the disjunction on the left-hand side. The last step of this proof is to admit that the inequality:

$$\sum_{i=1}^{K}A_l^{s_0}\left(\overrightarrow{\mathbf{P}_i}\right)\cdot A_l^i \leq Prob^{\infty}\left(s_0,\ \mathcal{G}\right) \leq \sum_{i=1}^{K}A_r^{s_0}\left(\overrightarrow{\mathbf{P}_i'}\right)\cdot A_r^i,$$

is a consequence, see Equation (C.59) and (C.60), of the inequalities under the probability measure of the *c. i.* in Equations (6.26) and (6.27). Therefore, by Equation (C.58), we have that Equation (C.57) holds. □

Lemma 73 *For any values* $A, B, C, D, E, F \in \mathbb{R}_{\geq 0}$ *such that* $A \leq C \leq B \leq D$ *and* $E \leq F$*, the following inequalities hold:*

$$A + E \leq C + E \leq B + F \leq D + F, \quad \textit{and} \quad A \cdot E \leq C \cdot E \leq B \cdot F \leq D \cdot F.$$

Proof Considering the fact that $A, B, C, D, E, F \in \mathbb{R}_{\geq 0}$, notice that:

- From $A \leq C$ it follows $A + E \leq C + E$, from $B \leq D$ it follows $B + F \leq D + F$, from $C \leq B$ and $E \leq F$ it follows $C + E \leq B + F$ and therefore we have:

$$A + E \leq C + E \leq B + F \leq D + F.$$

- From $A \leq C$ it follows $A \cdot E \leq C \cdot E$, from $B \leq D$ it follows $B \cdot F \leq D \cdot F$, from $C \leq B$ and $E \leq F$ it follows $C \cdot E \leq B \cdot F$ and therefore we have:

$$A \cdot E \leq C \cdot E \leq B \cdot F \leq D \cdot F.$$

□

Theorem 74 (The *c. i.* of the error) *For the* c. i. *given by Equation (6.26), based on independently obtained samples, and the error bounds given by Equation (6.29), the following* c. i. *results:*

$$Prob\left(\sum_{i=1}^{K}\left(\widetilde{p}_i^{s_0} - \varepsilon_i\right)\cdot A_l^i \leq Prob^{\infty}\left(s_0,\ \mathcal{G}\right) \leq \sum_{i=1}^{K}\left(\widetilde{p}_i^{s_0} + \varepsilon_i\right)\cdot A_r^i\right) \succeq \prod_{i=1}^{K}\xi_i^s, \tag{C.61}$$

under the assumption that for all $i \in I_{\mathcal{G}} : \widetilde{p}_i^{s_0} - \varepsilon_i, \widetilde{p}_i^{s_0} + \varepsilon_i, A_l^i, A_r^i \in \mathbb{R}_{[0,1]}$.

Proof First let us notice that $I = \{1, \ldots, K\}$. Then using the fact that the *c. i.* given by Equations (6.26) are based on independently obtained samples we can conclude that:

$$Prob\left(\bigwedge_{i\in I}\left(A_l^i \leq \pi_i^g \leq A_r^i\right)\right) \succeq \prod_{i=1}^{K} \xi_i^s. \tag{C.62}$$

Since for all $i \in I_{\mathcal{G}} : \widetilde{p}_i^{s_0} - \varepsilon_i, \widetilde{p}_i^{s_0} + \varepsilon_i, A_l^i, A_r^i \in \mathbb{R}_{[0,1]}$ we have an implication:

$$\begin{aligned}&\left(\left(A_l^i \leq \pi_i^g \leq A_r^i\right) \wedge \left(\widetilde{p}_i^{s_0} - \varepsilon_i \leq p_i^{s_0} \leq \widetilde{p}_i^{s_0} + \varepsilon_i\right)\right) \Longrightarrow \\ &\quad \Longrightarrow \left(\left(\widetilde{p}_i^{s_0} - \varepsilon_i\right) \cdot A_l^i \leq p_i^{s_0} \cdot \pi_i^g \leq \left(\widetilde{p}_i^{s_0} + \varepsilon_i\right) \cdot A_r^i\right).\end{aligned} \tag{C.63}$$

Now it suffices to notice that:

$$\begin{aligned}&\left(\bigwedge_{i\in I}\left(\left(\widetilde{p}_i^{s_0} - \varepsilon_i\right) \cdot A_l^i \leq p_i^{s_0} \cdot \pi_i^g \leq \left(\widetilde{p}_i^{s_0} + \varepsilon_i\right) \cdot A_r^i\right)\right) \Longrightarrow \\ \Longrightarrow &\left(\sum_{i=1}^{K}\left(\widetilde{p}_i^{s_0} - \varepsilon_i\right) \cdot A_l^i \leq \sum_{i=1}^{K} p_i^{s_0} \cdot \pi_i^g \leq \sum_{i=1}^{K}\left(\widetilde{p}_i^{s_0} + \varepsilon_i\right) \cdot A_r^i\right),\end{aligned} \tag{C.64}$$

where the right-hand side gives the *c. i.* borders for $Prob^{\infty}\left(s_0,\ \mathcal{G}\right)$, see Equation (6.25), that is obtained by summing up all the inequalities from the disjunction on the left-hand side. The last step of this proof is to admit that the inequality:

$$\sum_{i=1}^{K}\left(\widetilde{p}_i^{s_0} - \varepsilon_i\right) \cdot A_l^i \leq Prob^{\infty}\left(s_0,\ \mathcal{G}\right) \leq \sum_{i=1}^{K}\left(\widetilde{p}_i^{s_0} + \varepsilon_i\right) \cdot A_r^i,$$

is a consequence, see Equation (C.63) and (C.64), of the inequalities under the probability measure of the *c. i.* in Equation (6.26). Therefore, by Equation (C.62), we have that Equation (C.61) holds. □